

MPTRAC

Generated by Doxygen 1.9.1

<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	4
4.1.2 Field Documentation	4
4.2 cache_t Struct Reference	5
4.2.1 Detailed Description	5
4.2.2 Field Documentation	5
4.3 clim_t Struct Reference	6
4.3.1 Detailed Description	8
4.3.2 Field Documentation	8
4.4 ctl_t Struct Reference	12
4.4.1 Detailed Description	21
4.4.2 Field Documentation	21
4.5 met_t Struct Reference	56
4.5.1 Detailed Description	58
4.5.2 Field Documentation	58
<b>5 File Documentation</b>	<b>64</b>
5.1 atm_conv.c File Reference	64
5.1.1 Detailed Description	65
5.1.2 Function Documentation	65
5.2 atm_conv.c	66
5.3 atm_dist.c File Reference	67
5.3.1 Detailed Description	67
5.3.2 Function Documentation	67
5.4 atm_dist.c	72
5.5 atm_init.c File Reference	77
5.5.1 Detailed Description	77
5.5.2 Function Documentation	77
5.6 atm_init.c	79
5.7 atm_select.c File Reference	80
5.7.1 Detailed Description	81
5.7.2 Function Documentation	81
5.8 atm_select.c	83
5.9 atm_split.c File Reference	85
5.9.1 Detailed Description	85

5.9.2 Function Documentation	85
5.10 atm_split.c	87
5.11 atm_stat.c File Reference	89
5.11.1 Detailed Description	90
5.11.2 Function Documentation	90
5.12 atm_stat.c	93
5.13 day2doy.c File Reference	96
5.13.1 Detailed Description	96
5.13.2 Function Documentation	96
5.14 day2doy.c	97
5.15 doy2day.c File Reference	97
5.15.1 Detailed Description	97
5.15.2 Function Documentation	97
5.16 doy2day.c	98
5.17 jsec2time.c File Reference	99
5.17.1 Detailed Description	99
5.17.2 Function Documentation	99
5.18 jsec2time.c	100
5.19 libtrac.c File Reference	100
5.19.1 Detailed Description	104
5.19.2 Function Documentation	104
5.20 libtrac.c	202
5.21 libtrac.h File Reference	279
5.21.1 Detailed Description	287
5.21.2 Macro Definition Documentation	287
5.21.3 Function Documentation	307
5.22 libtrac.h	406
5.23 met_conv.c File Reference	427
5.23.1 Detailed Description	427
5.23.2 Function Documentation	427
5.24 met_conv.c	428
5.25 met_lapse.c File Reference	429
5.25.1 Detailed Description	430
5.25.2 Macro Definition Documentation	430
5.25.3 Function Documentation	430
5.26 met_lapse.c	434
5.27 met_map.c File Reference	437
5.27.1 Detailed Description	437
5.27.2 Macro Definition Documentation	438
5.27.3 Function Documentation	438
5.28 met_map.c	442
5.29 met_prof.c File Reference	446

---

5.29.1 Detailed Description . . . . .	446
5.29.2 Macro Definition Documentation . . . . .	446
5.29.3 Function Documentation . . . . .	446
5.30 met_prof.c . . . . .	450
5.31 met_sample.c File Reference . . . . .	453
5.31.1 Detailed Description . . . . .	454
5.31.2 Function Documentation . . . . .	454
5.32 met_sample.c . . . . .	457
5.33 met_spec.c File Reference . . . . .	460
5.33.1 Detailed Description . . . . .	460
5.33.2 Macro Definition Documentation . . . . .	460
5.33.3 Function Documentation . . . . .	460
5.34 met_spec.c . . . . .	463
5.35 met_subgrid.c File Reference . . . . .	465
5.35.1 Detailed Description . . . . .	466
5.35.2 Function Documentation . . . . .	466
5.36 met_subgrid.c . . . . .	469
5.37 met_zm.c File Reference . . . . .	472
5.37.1 Detailed Description . . . . .	472
5.37.2 Macro Definition Documentation . . . . .	472
5.37.3 Function Documentation . . . . .	473
5.38 met_zm.c . . . . .	476
5.39 sedi.c File Reference . . . . .	480
5.39.1 Detailed Description . . . . .	480
5.39.2 Function Documentation . . . . .	480
5.40 sedi.c . . . . .	481
5.41 time2jsec.c File Reference . . . . .	482
5.41.1 Detailed Description . . . . .	482
5.41.2 Function Documentation . . . . .	482
5.42 time2jsec.c . . . . .	483
5.43 tnat.c File Reference . . . . .	483
5.43.1 Detailed Description . . . . .	484
5.43.2 Function Documentation . . . . .	484
5.44 tnat.c . . . . .	485
5.45 trac.c File Reference . . . . .	485
5.45.1 Detailed Description . . . . .	486
5.45.2 Function Documentation . . . . .	486
5.46 trac.c . . . . .	515
5.47 tropo.c File Reference . . . . .	538
5.47.1 Detailed Description . . . . .	538
5.47.2 Function Documentation . . . . .	538
5.48 tropo.c . . . . .	542

5.49 tropo_sample.c File Reference . . . . .	546
5.49.1 Detailed Description . . . . .	546
5.49.2 Macro Definition Documentation . . . . .	546
5.49.3 Function Documentation . . . . .	546
5.50 tropo_sample.c . . . . .	551
5.51 tropo_zm.c File Reference . . . . .	555
5.51.1 Detailed Description . . . . .	555
5.51.2 Macro Definition Documentation . . . . .	555
5.51.3 Function Documentation . . . . .	556
5.52 tropo_zm.c . . . . .	558
5.53 wind.c File Reference . . . . .	561
5.53.1 Detailed Description . . . . .	561
5.53.2 Function Documentation . . . . .	561
5.54 wind.c . . . . .	563
<b>Index</b>	<b>567</b>

## 1 Main Page

Massive-Parallel Trajectory Calculations (MPTRAC) is a Lagrangian particle dispersion model for the free 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>	
Atmospheric data	<b>3</b>
<b>cache_t</b>	
Cache data	<b>5</b>
<b>clim_t</b>	
Climatological data	<b>6</b>
<b>ctl_t</b>	
Control parameters	<b>12</b>
<b>met_t</b>	
Meteo data	<b>56</b>

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">atm_conv.c</a>	Convert file format of air parcel data files	64
<a href="#">atm_dist.c</a>	Calculate transport deviations of trajectories	67
<a href="#">atm_init.c</a>	Create atmospheric data file with initial air parcel positions	77
<a href="#">atm_select.c</a>	Extract subsets of air parcels from atmospheric data files	80
<a href="#">atm_split.c</a>	Split air parcels into a larger number of parcels	85
<a href="#">atm_stat.c</a>	Calculate air parcel statistics	89
<a href="#">day2doy.c</a>	Convert date to day of year	96
<a href="#">doy2day.c</a>	Convert day of year to date	97
<a href="#">jsec2time.c</a>	Convert Julian seconds to date	99
<a href="#">libtrac.c</a>	MPTRAC library definitions	100
<a href="#">libtrac.h</a>	MPTRAC library declarations	279
<a href="#">met_conv.c</a>	Convert file format of meteo data files	427
<a href="#">met_lapse.c</a>	Calculate lapse rate statistics	429
<a href="#">met_map.c</a>	Extract map from meteorological data	437
<a href="#">met_prof.c</a>	Extract vertical profile from meteorological data	446
<a href="#">met_sample.c</a>	Sample meteorological data at given geolocations	453
<a href="#">met_spec.c</a>	Spectral analysis of meteorological data	460
<a href="#">met_subgrid.c</a>	Calculate standard deviations of horizontal wind and vertical velocity	465

<a href="#">met_zm.c</a>	Extract zonal mean from meteorological data	472
<a href="#">sedi.c</a>	Calculate sedimentation velocity	480
<a href="#">time2jsec.c</a>	Convert date to Julian seconds	482
<a href="#">tnat.c</a>	Calculate PSC temperatures	483
<a href="#">trac.c</a>	Lagrangian particle dispersion model	485
<a href="#">tropo.c</a>	Create tropopause data set from meteorological data	538
<a href="#">tropo_sample.c</a>	Sample tropopause data set	546
<a href="#">tropo_zm.c</a>	Extract zonal mean of tropopause data set	555
<a href="#">wind.c</a>	Create meteorological data files with synthetic wind fields	561

## 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 [zeta](#) [NP]  
*Zeta [K].*
- 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 [1330](#) 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 [1333](#) of file [libtrac.h](#).

##### 4.1.2.2 `time` `double atm_t::time[NP]`

Time [s].

Definition at line [1336](#) of file [libtrac.h](#).

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

Pressure [hPa].

Definition at line [1339](#) of file [libtrac.h](#).

##### 4.1.2.4 `zeta` `double atm_t::zeta[NP]`

Zeta [K].

Definition at line [1342](#) of file [libtrac.h](#).

##### 4.1.2.5 `lon` `double atm_t::lon[NP]`

Longitude [deg].

Definition at line [1345](#) of file [libtrac.h](#).



#### 4.1.2.6 `lat` `double atm_t::lat` [NP]

Latitude [deg].

Definition at line 1348 of file `libtrac.h`.

#### 4.1.2.7 `q` `double atm_t::q` [NQ] [NP]

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

Definition at line 1351 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

- `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.*
- `float uvwp` [NP][3]  
*Wind perturbations [m/s].*

### 4.2.1 Detailed Description

Cache data.

Definition at line 1356 of file `libtrac.h`.

### 4.2.2 Field Documentation

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

Isosurface variables.

Definition at line [1359](#) of file [libtrac.h](#).

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

Isosurface balloon pressure [hPa].

Definition at line [1362](#) of file [libtrac.h](#).

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

Isosurface balloon time [s].

Definition at line [1365](#) of file [libtrac.h](#).

#### 4.2.2.4 **iso\_n** `int cache_t::iso_n`

Isosurface balloon number of data points.

Definition at line [1368](#) of file [libtrac.h](#).

#### 4.2.2.5 **uvwp** `float cache_t::uvwp[NP][3]`

Wind perturbations [m/s].

Definition at line [1371](#) of file [libtrac.h](#).

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

- [libtrac.h](#)

### 4.3 **clim\_t** Struct Reference

Climatological data.

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

## Data Fields

- int [tropo\\_ftime](#)  
*Number of tropopause timesteps.*
- int [tropo\\_nlat](#)  
*Number of tropopause latitudes.*
- double [tropo\\_time](#) [12]  
*Tropopause time steps [s].*
- double [tropo\\_lat](#) [73]  
*Tropopause latitudes [deg].*
- double [tropo](#) [12][73]  
*Tropopause pressure values [hPa].*
- int [hno3\\_ftime](#)  
*Number of HNO3 timesteps.*
- int [hno3\\_nlat](#)  
*Number of HNO3 latitudes.*
- int [hno3\\_np](#)  
*Number of HNO3 pressure levels.*
- double [hno3\\_time](#) [12]  
*HNO3 time steps [s].*
- double [hno3\\_lat](#) [18]  
*HNO3 latitudes [deg].*
- double [hno3\\_p](#) [10]  
*HNO3 pressure levels [hPa].*
- double [hno3](#) [12][18][10]  
*HNO3 volume mixing ratios [ppv].*
- int [oh\\_ftime](#)  
*Number of OH timesteps.*
- int [oh\\_nlat](#)  
*Number of OH latitudes.*
- int [oh\\_np](#)  
*Number of OH pressure levels.*
- double [oh\\_time](#) [CT]  
*OH time steps [s].*
- double [oh\\_lat](#) [CY]  
*OH latitudes [deg].*
- double [oh\\_p](#) [CP]  
*OH pressure levels [hPa].*
- double [oh](#) [CT][CP][CY]  
*OH number concentrations [molec/cm<sup>3</sup>].*
- int [h2o2\\_ftime](#)  
*Number of H2O2 timesteps.*
- int [h2o2\\_nlat](#)  
*Number of H2O2 latitudes.*
- int [h2o2\\_np](#)  
*Number of H2O2 pressure levels.*
- double [h2o2\\_time](#) [CT]  
*H2O2 time steps [s].*
- double [h2o2\\_lat](#) [CY]  
*H2O2 latitudes [deg].*
- double [h2o2\\_p](#) [CP]  
*H2O2 pressure levels [hPa].*
- double [h2o2](#) [CT][CP][CY]  
*H2O2 number concentrations [molec/cm<sup>3</sup>].*

#### 4.3.1 Detailed Description

Climatological data.

Definition at line [1376](#) of file [libtrac.h](#).

#### 4.3.2 Field Documentation

##### 4.3.2.1 **tropo\_ntime** `int clim_t::tropo_ntime`

Number of tropopause timesteps.

Definition at line [1379](#) of file [libtrac.h](#).

##### 4.3.2.2 **tropo\_nlat** `int clim_t::tropo_nlat`

Number of tropopause latitudes.

Definition at line [1382](#) of file [libtrac.h](#).

##### 4.3.2.3 **tropo\_time** `double clim_t::tropo_time[12]`

Tropopause time steps [s].

Definition at line [1385](#) of file [libtrac.h](#).

##### 4.3.2.4 **tropo\_lat** `double clim_t::tropo_lat[73]`

Tropopause latitudes [deg].

Definition at line [1388](#) of file [libtrac.h](#).

##### 4.3.2.5 **tropo** `double clim_t::tropo[12][73]`

Tropopause pressure values [hPa].

Definition at line [1391](#) of file [libtrac.h](#).

**4.3.2.6 hno3\_ntime** `int clim_t::hno3_ntime`

Number of HNO3 timesteps.

Definition at line [1394](#) of file [libtrac.h](#).

**4.3.2.7 hno3\_nlat** `int clim_t::hno3_nlat`

Number of HNO3 latitudes.

Definition at line [1397](#) of file [libtrac.h](#).

**4.3.2.8 hno3\_np** `int clim_t::hno3_np`

Number of HNO3 pressure levels.

Definition at line [1400](#) of file [libtrac.h](#).

**4.3.2.9 hno3\_time** `double clim_t::hno3_time[12]`

HNO3 time steps [s].

Definition at line [1403](#) of file [libtrac.h](#).

**4.3.2.10 hno3\_lat** `double clim_t::hno3_lat[18]`

HNO3 latitudes [deg].

Definition at line [1406](#) of file [libtrac.h](#).

**4.3.2.11 hno3\_p** `double clim_t::hno3_p[10]`

HNO3 pressure levels [hPa].

Definition at line [1409](#) of file [libtrac.h](#).

**4.3.2.12 hno3** `double clim_t::hno3[12][18][10]`

HNO3 volume mixing ratios [ppv].

Definition at line [1412](#) of file [libtrac.h](#).

**4.3.2.13 oh\_ntime** `int clim_t::oh_ntime`

Number of OH timesteps.

Definition at line [1415](#) of file [libtrac.h](#).

**4.3.2.14 oh\_nlat** `int clim_t::oh_nlat`

Number of OH latitudes.

Definition at line [1418](#) of file [libtrac.h](#).

**4.3.2.15 oh\_np** `int clim_t::oh_np`

Number of OH pressure levels.

Definition at line [1421](#) of file [libtrac.h](#).

**4.3.2.16 oh\_time** `double clim_t::oh_time[CT]`

OH time steps [s].

Definition at line [1424](#) of file [libtrac.h](#).

**4.3.2.17 oh\_lat** `double clim_t::oh_lat[CY]`

OH latitudes [deg].

Definition at line [1427](#) of file [libtrac.h](#).

**4.3.2.18 oh\_p** double clim\_t::oh\_p[CP]

OH pressure levels [hPa].

Definition at line 1430 of file libtrac.h.

**4.3.2.19 oh** double clim\_t::oh[CT][CP][CY]

OH number concentrations [molec/cm<sup>3</sup>].

Definition at line 1433 of file libtrac.h.

**4.3.2.20 h2o2\_ntime** int clim\_t::h2o2\_ntime

Number of H2O2 timesteps.

Definition at line 1436 of file libtrac.h.

**4.3.2.21 h2o2\_nlat** int clim\_t::h2o2\_nlat

Number of H2O2 latitudes.

Definition at line 1439 of file libtrac.h.

**4.3.2.22 h2o2\_np** int clim\_t::h2o2\_np

Number of H2O2 pressure levels.

Definition at line 1442 of file libtrac.h.

**4.3.2.23 h2o2\_time** double clim\_t::h2o2\_time[CT]

H2O2 time steps [s].

Definition at line 1445 of file libtrac.h.

**4.3.2.24 h2o2\_lat** `double clim_t::h2o2_lat[CY]`

H2O2 latitudes [deg].

Definition at line 1448 of file [libtrac.h](#).

**4.3.2.25 h2o2\_p** `double clim_t::h2o2_p[CP]`

H2O2 pressure levels [hPa].

Definition at line 1451 of file [libtrac.h](#).

**4.3.2.26 h2o2** `double clim_t::h2o2[CT][CP][CY]`

H2O2 number concentrations [molec/cm<sup>3</sup>].

Definition at line 1454 of file [libtrac.h](#).

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

- [libtrac.h](#)

## 4.4 ctl\_t Struct Reference

Control parameters.

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

### Data Fields

- int [vert\\_coord\\_ap](#)  
*Vertical coordinate of air parcels (0=pressure, 1=zeta).*
- int [vert\\_coord\\_met](#)  
*Vertical coordinate of input meteo data (0=automatic, 1=eta).*
- int [vert\\_vel](#)  
*Vertical velocity (0=kinematic, 1=diabatic).*
- int [clams\\_met\\_data](#)  
*Read MPTRAC or CLaMS meteo data (0=MPTRAC, 1=CLaMS).*
- size\_t [chunkszhint](#)  
*Chunk size hint for nc\_\_open.*
- int [read\\_mode](#)  
*Read mode for nc\_\_open.*
- int [nq](#)  
*Number of quantities.*
- char [qnt\\_name](#) [NQ][LEN]  
*Quantity names.*



- char `qnt_longname` [NQ][LEN]  
*Quantity long names.*
- char `qnt_unit` [NQ][LEN]  
*Quantity units.*
- char `qnt_format` [NQ][LEN]  
*Quantity output format.*
- int `qnt_idx`  
*Quantity array index for air parcel IDs.*
- int `qnt_ens`  
*Quantity array index for ensemble IDs.*
- int `qnt_stat`  
*Quantity array index for station flag.*
- int `qnt_m`  
*Quantity array index for mass.*
- int `qnt_vmr`  
*Quantity array index for volume mixing ratio.*
- int `qnt_rp`  
*Quantity array index for particle radius.*
- int `qnt_rhop`  
*Quantity array index for particle density.*
- 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_pbl`  
*Quantity array index for boundary layer pressure.*
- 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_rho`  
*Quantity array index for density of air.*
- 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\\_pct](#)
  - Quantity array index for cloud top pressure.*
- int [qnt\\_pcb](#)
  - Quantity array index for cloud bottom 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\\_cin](#)
  - Quantity array index for convective inhibition (CIN).*
- int [qnt\\_hno3](#)
  - Quantity array index for nitric acid vmr.*
- int [qnt\\_oh](#)
  - Quantity array index for hydroxyl number concentrations.*
- int [qnt\\_vmrimpl](#)
  - Quantity array index for implicity volumn mixing ratio.*
- int [qnt\\_mloss\\_oh](#)
  - Quantity array index for total mass loss due to OH chemistry.*
- int [qnt\\_mloss\\_h2o2](#)
  - Quantity array index for total mass loss due to H2O2 chemistry.*
- int [qnt\\_mloss\\_wet](#)
  - Quantity array index for total mass loss due to wet deposition.*
- int [qnt\\_mloss\\_dry](#)
  - Quantity array index for total mass loss due to dry deposition.*
- int [qnt\\_mloss\\_decay](#)
  - Quantity array index for total mass loss due to exponential decax.*
- 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_zeta`  
*Quantity array index for zeta vertical coordinate.*
- 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 `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 meteo data.*
- double `dt_met`  
*Time step of meteo data [s].*
- int `met_type`  
*Type of meteo data files (0=netCDF, 1=binary, 2=pack, 3=zfp, 4=zstd).*
- int `met_nc_scale`  
*Check netCDF scaling factors (0=no, 1=yes).*
- 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, 5=dynamical).*
- double `met_tropo_lapse`

*WMO tropopause lapse rate [K/km].*
- int `met_tropo_nlev`

*WMO tropopause layer depth (number of levels).*
- double `met_tropo_lapse_sep`

*WMO tropopause separation layer lapse rate [K/km].*
- int `met_tropo_nlev_sep`

*WMO tropopause separation layer depth (number of levels).*
- double `met_tropo_pv`

*Dynamical tropopause potential vorticity threshold [PVU].*
- double `met_tropo_theta`

*Dynamical tropopause potential temperature threshold [K].*
- int `met_tropo_spline`

*Tropopause interpolation method (0=linear, 1=spline).*
- int `met_cloud`

*Cloud data (0=none, 1=LWC+IWC, 2=RW+SWC, 3=all).*
- double `met_cloud_min`

*Minimum cloud ice water content [kg/kg].*
- double `met_dt_out`

*Time step for sampling of meteo data along trajectories [s].*
- int `met_cache`

*Preload meteo data into disk cache (0=no, 1=yes).*
- double `sort_dt`

*Time step for sorting of particle data [s].*
- int `isosurf`

*Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).*
- char `balloon` [LEN]

*Balloon position filename.*
- int `advect`

*Advection scheme (1=Euler, 2=midpoint, 4=Runge-Kutta).*
- int `reflect`

*Reflection of particles at top and bottom boundary (0=no, 1=yes).*
- double `turb_dx_trop`

*Horizontal turbulent diffusion coefficient (troposphere) [ $m^2/s$ ].*
- double `turb_dx_strat`

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

- double `turb_dz_trop`  
*Vertical turbulent diffusion coefficient (troposphere) [ $m^2/s$ ].*
- double `turb_dz_strat`  
*Vertical turbulent diffusion coefficient (stratosphere) [ $m^2/s$ ].*
- 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].*
- double `conv_cin`  
*CIN threshold for convection module [J/kg].*
- double `conv_wmax`  
*Maximum vertical velocity for convection module [m/s].*
- double `conv_wcape`  
*Limit vertical velocity based on CAPE (0=no, 1=yes).*
- double `conv_dt`  
*Time interval for convection module [s].*
- int `conv_mix`  
*Type of vertical mixing (0=pressure, 1=density).*
- int `conv_mix_bot`  
*Lower level for mixing (0=particle pressure, 1=surface).*
- int `conv_mix_top`  
*Upper level for mixing (0=particle pressure, 1=EL).*
- double `bound_mass`  
*Boundary conditions mass per particle [kg].*
- double `bound_mass_trend`  
*Boundary conditions mass per particle trend [kg/s].*
- double `bound_vmr`  
*Boundary conditions volume mixing ratio [ppv].*
- double `bound_vmr_trend`  
*Boundary conditions volume mixing ratio trend [ppv/s].*
- double `bound_lat0`  
*Boundary conditions minimum longitude [deg].*
- double `bound_lat1`  
*Boundary conditions maximum longitude [deg].*
- double `bound_p0`  
*Boundary conditions bottom pressure [hPa].*
- double `bound_p1`  
*Boundary conditions top pressure [hPa].*
- double `bound_dps`  
*Boundary conditions delta to surface pressure [hPa].*
- 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].*
- char `clim_oh_filename` [LEN]

- Filename of OH climatology.
  - char `clim_h2o2_filename` [LEN]
- Filename of H2O2 climatology.
  - int `oh_chem_reaction`
- Reaction type for OH chemistry (0=none, 2=bimolecular, 3=termolecular).
  - double `oh_chem` [4]
- Coefficients for OH reaction rate (A, E/R or k0, n, kinf, m).
  - double `oh_chem_beta`
- Beta parameter for diurnal variability of OH.
  - double `h2o2_chem_cc`
- Cloud cover parameter for H2O2 chemistry.
  - int `h2o2_chem_reaction`
- Reaction type for H2O2 chemistry (0=none, 1=SO2).
  - double `dry_depo` [1]
- Coefficients for dry deposition (v).
  - double `wet_depo_pre` [2]
- Coefficients for precipitation calculation.
  - double `wet_depo_bc_a`
- Coefficient A for wet deposition below cloud (exponential form).
  - double `wet_depo_bc_b`
- Coefficient B for wet deposition below cloud (exponential form).
  - double `wet_depo_ic_a`
- Coefficient A for wet deposition in cloud (exponential form).
  - double `wet_depo_ic_b`
- Coefficient B for wet deposition in cloud (exponential form).
  - double `wet_depo_ic_h` [3]
- Coefficients for wet deposition in cloud (Henry's law: Hb, Cb, pH).
  - double `wet_depo_bc_h` [2]
- Coefficients for wet deposition below cloud (Henry's law: Hb, Cb).
  - double `wet_depo_ic_ret_ratio`
- Coefficients for wet deposition in cloud: retention ratio.
  - double `wet_depo_bc_ret_ratio`
- Coefficients for wet deposition below cloud: retention ratio.
  - double `psc_h2o`
- H2O volume mixing ratio for PSC analysis.
  - double `psc_hno3`
- HNO3 volume mixing ratio for PSC analysis.
  - char `atm_basename` [LEN]
- Basename of atmospheric data files.
  - char `atm_gpfile` [LEN]
- Gnuplot file for atmospheric data.
  - double `atm_dt_out`
- Time step for atmospheric data output [s].
  - int `atm_filter`
- Time filter for atmospheric data output (0=none, 1=missval, 2=remove).
  - int `atm_stride`
- Particle index stride for atmospheric data files.
  - int `atm_type`
- Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF, 3=CLaMS).
  - char `csi_basename` [LEN]
- Basename of CSI data files.

- double `csi_dt_out`  
*Time step for CSI data output [s].*
- char `csi_obsfile` [LEN]  
*Observation data file for CSI analysis.*
- double `csi_obsmin`  
*Minimum observation index to trigger detection.*
- double `csi_modmin`  
*Minimum column density to trigger detection [kg/m<sup>2</sup>].*
- int `csi_nz`  
*Number of altitudes of gridded CSI data.*
- double `csi_z0`  
*Lower altitude of gridded CSI data [km].*
- double `csi_z1`  
*Upper altitude of gridded CSI data [km].*
- int `csi_nx`  
*Number of longitudes of gridded CSI data.*
- double `csi_lon0`  
*Lower longitude of gridded CSI data [deg].*
- double `csi_lon1`  
*Upper longitude of gridded CSI data [deg].*
- int `csi_ny`  
*Number of latitudes of gridded CSI data.*
- double `csi_lat0`  
*Lower latitude of gridded CSI data [deg].*
- double `csi_lat1`  
*Upper latitude of gridded CSI data [deg].*
- char `ens_basename` [LEN]  
*Basename of ensemble data file.*
- double `ens_dt_out`  
*Time step for ensemble output [s].*
- char `grid_basename` [LEN]  
*Basename of grid data files.*
- char `grid_gpfile` [LEN]  
*Gnuplot file for gridded data.*
- double `grid_dt_out`  
*Time step for gridded data output [s].*
- int `grid_sparse`  
*Sparse output in grid data files (0=no, 1=yes).*
- int `grid_nz`  
*Number of altitudes of gridded data.*
- double `grid_z0`  
*Lower altitude of gridded data [km].*
- double `grid_z1`  
*Upper altitude of gridded data [km].*
- int `grid_nx`  
*Number of longitudes of gridded data.*
- double `grid_lon0`  
*Lower longitude of gridded data [deg].*
- double `grid_lon1`  
*Upper longitude of gridded data [deg].*
- int `grid_ny`

- Number of latitudes of gridded data.*
- double `grid_lat0`  
*Lower latitude of gridded data [deg].*
- double `grid_lat1`  
*Upper latitude of gridded data [deg].*
- int `grid_type`  
*Type of grid data files (0=ASCII, 1=netCDF).*
- 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 `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 depth 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].*
- double `stat_t0`  
*Start time for station output [s].*
- double `stat_t1`  
*Stop time for station output [s].*



#### 4.4.1 Detailed Description

Control parameters.

Definition at line 690 of file [libtrac.h](#).

#### 4.4.2 Field Documentation

##### 4.4.2.1 `vert_coord_ap` `int ctl_t::vert_coord_ap`

Vertical coordinate of air parcels (0=pressure, 1=zeta).

Definition at line 693 of file [libtrac.h](#).

##### 4.4.2.2 `vert_coord_met` `int ctl_t::vert_coord_met`

Vertical coordinate of input meteo data (0=automatic, 1=eta).

Definition at line 696 of file [libtrac.h](#).

##### 4.4.2.3 `vert_vel` `int ctl_t::vert_vel`

Vertical velocity (0=kinematic, 1=diabatic).

Definition at line 699 of file [libtrac.h](#).

##### 4.4.2.4 `clams_met_data` `int ctl_t::clams_met_data`

Read MPTRAC or CLaMS meteo data (0=MPTRAC, 1=CLaMS).

Definition at line 702 of file [libtrac.h](#).

##### 4.4.2.5 `chunkszhint` `size_t ctl_t::chunkszhint`

Chunk size hint for `nc__open`.

Definition at line 705 of file [libtrac.h](#).

**4.4.2.6 read\_mode** `int ctl_t::read_mode`

Read mode for nc\_\_open.

Definition at line [708](#) of file [libtrac.h](#).

**4.4.2.7 nq** `int ctl_t::nq`

Number of quantities.

Definition at line [711](#) of file [libtrac.h](#).

**4.4.2.8 qnt\_name** `char ctl_t::qnt_name[NQ][LEN]`

Quantity names.

Definition at line [714](#) of file [libtrac.h](#).

**4.4.2.9 qnt\_longname** `char ctl_t::qnt_longname[NQ][LEN]`

Quantity long names.

Definition at line [717](#) of file [libtrac.h](#).

**4.4.2.10 qnt\_unit** `char ctl_t::qnt_unit[NQ][LEN]`

Quantity units.

Definition at line [720](#) of file [libtrac.h](#).

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

Quantity output format.

Definition at line [723](#) of file [libtrac.h](#).

**4.4.2.12** `qnt_idx` `int` `ctl_t::qnt_idx`

Quantity array index for air parcel IDs.

Definition at line 726 of file [libtrac.h](#).

**4.4.2.13** `qnt_ens` `int` `ctl_t::qnt_ens`

Quantity array index for ensemble IDs.

Definition at line 729 of file [libtrac.h](#).

**4.4.2.14** `qnt_stat` `int` `ctl_t::qnt_stat`

Quantity array index for station flag.

Definition at line 732 of file [libtrac.h](#).

**4.4.2.15** `qnt_m` `int` `ctl_t::qnt_m`

Quantity array index for mass.

Definition at line 735 of file [libtrac.h](#).

**4.4.2.16** `qnt_vmr` `int` `ctl_t::qnt_vmr`

Quantity array index for volume mixing ratio.

Definition at line 738 of file [libtrac.h](#).

**4.4.2.17** `qnt_rp` `int` `ctl_t::qnt_rp`

Quantity array index for particle radius.

Definition at line 741 of file [libtrac.h](#).

**4.4.2.18 qnt\_rhop** `int ctl_t::qnt_rhop`

Quantity array index for particle density.

Definition at line [744](#) of file [libtrac.h](#).

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

Quantity array index for surface pressure.

Definition at line [747](#) of file [libtrac.h](#).

**4.4.2.20 qnt\_ts** `int ctl_t::qnt_ts`

Quantity array index for surface temperature.

Definition at line [750](#) of file [libtrac.h](#).

**4.4.2.21 qnt\_zs** `int ctl_t::qnt_zs`

Quantity array index for surface geopotential height.

Definition at line [753](#) of file [libtrac.h](#).

**4.4.2.22 qnt\_us** `int ctl_t::qnt_us`

Quantity array index for surface zonal wind.

Definition at line [756](#) of file [libtrac.h](#).

**4.4.2.23 qnt\_vs** `int ctl_t::qnt_vs`

Quantity array index for surface meridional wind.

Definition at line [759](#) of file [libtrac.h](#).

**4.4.2.24** `qnt_pbl` `int` `ctl_t::qnt_pbl`

Quantity array index for boundary layer pressure.

Definition at line 762 of file [libtrac.h](#).

**4.4.2.25** `qnt_pt` `int` `ctl_t::qnt_pt`

Quantity array index for tropopause pressure.

Definition at line 765 of file [libtrac.h](#).

**4.4.2.26** `qnt_tt` `int` `ctl_t::qnt_tt`

Quantity array index for tropopause temperature.

Definition at line 768 of file [libtrac.h](#).

**4.4.2.27** `qnt_zt` `int` `ctl_t::qnt_zt`

Quantity array index for tropopause geopotential height.

Definition at line 771 of file [libtrac.h](#).

**4.4.2.28** `qnt_h2ot` `int` `ctl_t::qnt_h2ot`

Quantity array index for tropopause water vapor vmr.

Definition at line 774 of file [libtrac.h](#).

**4.4.2.29** `qnt_z` `int` `ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line 777 of file [libtrac.h](#).

**4.4.2.30** `qnt_p` `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line [780](#) of file [libtrac.h](#).

**4.4.2.31** `qnt_t` `int ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line [783](#) of file [libtrac.h](#).

**4.4.2.32** `qnt_rho` `int ctl_t::qnt_rho`

Quantity array index for density of air.

Definition at line [786](#) of file [libtrac.h](#).

**4.4.2.33** `qnt_u` `int ctl_t::qnt_u`

Quantity array index for zonal wind.

Definition at line [789](#) of file [libtrac.h](#).

**4.4.2.34** `qnt_v` `int ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line [792](#) of file [libtrac.h](#).

**4.4.2.35** `qnt_w` `int ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line [795](#) of file [libtrac.h](#).

**4.4.2.36** `qnt_h2o` `int` `ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line 798 of file [libtrac.h](#).

**4.4.2.37** `qnt_o3` `int` `ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line 801 of file [libtrac.h](#).

**4.4.2.38** `qnt_lwc` `int` `ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line 804 of file [libtrac.h](#).

**4.4.2.39** `qnt_iwc` `int` `ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line 807 of file [libtrac.h](#).

**4.4.2.40** `qnt_pct` `int` `ctl_t::qnt_pct`

Quantity array index for cloud top pressure.

Definition at line 810 of file [libtrac.h](#).

**4.4.2.41** `qnt_pcb` `int` `ctl_t::qnt_pcb`

Quantity array index for cloud bottom pressure.

Definition at line 813 of file [libtrac.h](#).

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

Quantity array index for total column cloud water.

Definition at line [816](#) of file [libtrac.h](#).

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

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

Definition at line [819](#) of file [libtrac.h](#).

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

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

Definition at line [822](#) of file [libtrac.h](#).

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

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

Definition at line [825](#) of file [libtrac.h](#).

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

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

Definition at line [828](#) of file [libtrac.h](#).

**4.4.2.47 qnt\_cin** `int ctl_t::qnt_cin`

Quantity array index for convective inhibition (CIN).

Definition at line [831](#) of file [libtrac.h](#).



**4.4.2.48** `qnt_hno3` `int ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line 834 of file [libtrac.h](#).

**4.4.2.49** `qnt_oh` `int ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line 837 of file [libtrac.h](#).

**4.4.2.50** `qnt_vmrimpl` `int ctl_t::qnt_vmrimpl`

Quantity array index for implicit volume mixing ratio.

Definition at line 840 of file [libtrac.h](#).

**4.4.2.51** `qnt_mloss_oh` `int ctl_t::qnt_mloss_oh`

Quantity array index for total mass loss due to OH chemistry.

Definition at line 843 of file [libtrac.h](#).

**4.4.2.52** `qnt_mloss_h2o2` `int ctl_t::qnt_mloss_h2o2`

Quantity array index for total mass loss due to H2O2 chemistry.

Definition at line 846 of file [libtrac.h](#).

**4.4.2.53** `qnt_mloss_wet` `int ctl_t::qnt_mloss_wet`

Quantity array index for total mass loss due to wet deposition.

Definition at line 849 of file [libtrac.h](#).

**4.4.2.54 qnt\_mloss\_dry** `int ctl_t::qnt_mloss_dry`

Quantity array index for total mass loss due to dry deposition.

Definition at line [852](#) of file [libtrac.h](#).

**4.4.2.55 qnt\_mloss\_decay** `int ctl_t::qnt_mloss_decay`

Quantity array index for total mass loss due to exponential decax.

Definition at line [855](#) of file [libtrac.h](#).

**4.4.2.56 qnt\_psat** `int ctl_t::qnt_psat`

Quantity array index for saturation pressure over water.

Definition at line [858](#) of file [libtrac.h](#).

**4.4.2.57 qnt\_psice** `int ctl_t::qnt_psice`

Quantity array index for saturation pressure over ice.

Definition at line [861](#) of file [libtrac.h](#).

**4.4.2.58 qnt\_pw** `int ctl_t::qnt_pw`

Quantity array index for partial water vapor pressure.

Definition at line [864](#) of file [libtrac.h](#).

**4.4.2.59 qnt\_sh** `int ctl_t::qnt_sh`

Quantity array index for specific humidity.

Definition at line [867](#) of file [libtrac.h](#).

**4.4.2.60** `qnt_rh` `int ctl_t::qnt_rh`

Quantity array index for relative humidity over water.

Definition at line 870 of file [libtrac.h](#).

**4.4.2.61** `qnt_rhice` `int ctl_t::qnt_rhice`

Quantity array index for relative humidity over ice.

Definition at line 873 of file [libtrac.h](#).

**4.4.2.62** `qnt_theta` `int ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line 876 of file [libtrac.h](#).

**4.4.2.63** `qnt_zeta` `int ctl_t::qnt_zeta`

Quantity array index for zeta vertical coordinate.

Definition at line 879 of file [libtrac.h](#).

**4.4.2.64** `qnt_tvirt` `int ctl_t::qnt_tvirt`

Quantity array index for virtual temperature.

Definition at line 882 of file [libtrac.h](#).

**4.4.2.65** `qnt_lapse` `int ctl_t::qnt_lapse`

Quantity array index for lapse rate.

Definition at line 885 of file [libtrac.h](#).

**4.4.2.66** `qnt_vh` `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line [888](#) of file [libtrac.h](#).

**4.4.2.67** `qnt_vz` `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line [891](#) of file [libtrac.h](#).

**4.4.2.68** `qnt_pv` `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line [894](#) of file [libtrac.h](#).

**4.4.2.69** `qnt_tdew` `int ctl_t::qnt_tdew`

Quantity array index for dew point temperature.

Definition at line [897](#) of file [libtrac.h](#).

**4.4.2.70** `qnt_tice` `int ctl_t::qnt_tice`

Quantity array index for T\_ice.

Definition at line [900](#) of file [libtrac.h](#).

**4.4.2.71** `qnt_tsts` `int ctl_t::qnt_tsts`

Quantity array index for T\_STS.

Definition at line [903](#) of file [libtrac.h](#).

**4.4.2.72** `qnt_tnat` `int` `ctl_t::qnt_tnat`

Quantity array index for T\_NAT.

Definition at line 906 of file [libtrac.h](#).

**4.4.2.73** `direction` `int` `ctl_t::direction`

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

Definition at line 909 of file [libtrac.h](#).

**4.4.2.74** `t_start` `double` `ctl_t::t_start`

Start time of simulation [s].

Definition at line 912 of file [libtrac.h](#).

**4.4.2.75** `t_stop` `double` `ctl_t::t_stop`

Stop time of simulation [s].

Definition at line 915 of file [libtrac.h](#).

**4.4.2.76** `dt_mod` `double` `ctl_t::dt_mod`

Time step of simulation [s].

Definition at line 918 of file [libtrac.h](#).

**4.4.2.77** `metbase` `char` `ctl_t::metbase`[[LEN](#)]

Basename for meteo data.

Definition at line 921 of file [libtrac.h](#).

**4.4.2.78 dt\_met** `double ctl_t::dt_met`

Time step of meteo data [s].

Definition at line [924](#) of file [libtrac.h](#).

**4.4.2.79 met\_type** `int ctl_t::met_type`

Type of meteo data files (0=netCDF, 1=binary, 2=pack, 3=zfp, 4=zstd).

Definition at line [927](#) of file [libtrac.h](#).

**4.4.2.80 met\_nc\_scale** `int ctl_t::met_nc_scale`

Check netCDF scaling factors (0=no, 1=yes).

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

**4.4.2.81 met\_dx** `int ctl_t::met_dx`

Stride for longitudes.

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

**4.4.2.82 met\_dy** `int ctl_t::met_dy`

Stride for latitudes.

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

**4.4.2.83 met\_dp** `int ctl_t::met_dp`

Stride for pressure levels.

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

**4.4.2.84** `met_sx` `int` `ctl_t::met_sx`

Smoothing for longitudes.

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

**4.4.2.85** `met_sy` `int` `ctl_t::met_sy`

Smoothing for latitudes.

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

**4.4.2.86** `met_sp` `int` `ctl_t::met_sp`

Smoothing for pressure levels.

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

**4.4.2.87** `met_detrend` `double` `ctl_t::met_detrend`

FWHM of horizontal Gaussian used for detrending [km].

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

**4.4.2.88** `met_np` `int` `ctl_t::met_np`

Number of target pressure levels.

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

**4.4.2.89** `met_p` `double` `ctl_t::met_p` [\[EP\]](#)

Target pressure levels [hPa].

Definition at line 957 of file [libtrac.h](#).

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

Longitudinal smoothing of geopotential heights.

Definition at line 960 of file [libtrac.h](#).

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

Latitudinal smoothing of geopotential heights.

Definition at line 963 of file [libtrac.h](#).

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

Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO\_1st, 4=WMO\_2nd, 5=dynamical).

Definition at line 967 of file [libtrac.h](#).

**4.4.2.93 met\_tropo\_lapse** `double ctl_t::met_tropo_lapse`

WMO tropopause lapse rate [K/km].

Definition at line 970 of file [libtrac.h](#).

**4.4.2.94 met\_tropo\_nlev** `int ctl_t::met_tropo_nlev`

WMO tropopause layer depth (number of levels).

Definition at line 973 of file [libtrac.h](#).

**4.4.2.95 met\_tropo\_lapse\_sep** `double ctl_t::met_tropo_lapse_sep`

WMO tropopause separation layer lapse rate [K/km].

Definition at line 976 of file [libtrac.h](#).



**4.4.2.96 `met_tropo_nlev_sep`** `int ctl_t::met_tropo_nlev_sep`

WMO tropopause separation layer depth (number of levels).

Definition at line 979 of file [libtrac.h](#).

**4.4.2.97 `met_tropo_pv`** `double ctl_t::met_tropo_pv`

Dynamical tropopause potential vorticity threshold [PVU].

Definition at line 982 of file [libtrac.h](#).

**4.4.2.98 `met_tropo_theta`** `double ctl_t::met_tropo_theta`

Dynamical tropopause potential temperature threshold [K].

Definition at line 985 of file [libtrac.h](#).

**4.4.2.99 `met_tropo_spline`** `int ctl_t::met_tropo_spline`

Tropopause interpolation method (0=linear, 1=spline).

Definition at line 988 of file [libtrac.h](#).

**4.4.2.100 `met_cloud`** `int ctl_t::met_cloud`

Cloud data (0=none, 1=LWC+IWC, 2=RWC+SWC, 3=all).

Definition at line 991 of file [libtrac.h](#).

**4.4.2.101 `met_cloud_min`** `double ctl_t::met_cloud_min`

Minimum cloud ice water content [kg/kg].

Definition at line 994 of file [libtrac.h](#).

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

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

Definition at line [997](#) of file [libtrac.h](#).

**4.4.2.103 met\_cache** `int ctl_t::met_cache`

Preload meteo data into disk cache (0=no, 1=yes).

Definition at line [1000](#) of file [libtrac.h](#).

**4.4.2.104 sort\_dt** `double ctl_t::sort_dt`

Time step for sorting of particle data [s].

Definition at line [1003](#) of file [libtrac.h](#).

**4.4.2.105 isosurf** `int ctl_t::isosurf`

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

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

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

Balloon position filename.

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

**4.4.2.107 advect** `int ctl_t::advect`

Advection scheme (1=Euler, 2=midpoint, 4=Runge-Kutta).

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

**4.4.2.108** `reflect` `int ctl_t::reflect`

Reflection of particles at top and bottom boundary (0=no, 1=yes).

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

**4.4.2.109** `turb_dx_trop` `double ctl_t::turb_dx_trop`

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

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

**4.4.2.110** `turb_dx_strat` `double ctl_t::turb_dx_strat`

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

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

**4.4.2.111** `turb_dz_trop` `double ctl_t::turb_dz_trop`

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

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

**4.4.2.112** `turb_dz_strat` `double ctl_t::turb_dz_strat`

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

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

**4.4.2.113** `turb_mesox` `double ctl_t::turb_mesox`

Horizontal scaling factor for mesoscale wind fluctuations.

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

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

Vertical scaling factor for mesoscale wind fluctuations.

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

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

CAPE threshold for convection module [J/kg].

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

**4.4.2.116 conv\_cin** `double ctl_t::conv_cin`

CIN threshold for convection module [J/kg].

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

**4.4.2.117 conv\_wmax** `double ctl_t::conv_wmax`

Maximum vertical velocity for convection module [m/s].

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

**4.4.2.118 conv\_wcape** `double ctl_t::conv_wcape`

Limit vertical velocity based on CAPE (0=no, 1=yes).

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

**4.4.2.119 conv\_dt** `double ctl_t::conv_dt`

Time interval for convection module [s].

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

**4.4.2.120 `conv_mix`** `int ctl_t::conv_mix`

Type of vertical mixing (0=pressure, 1=density).

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

**4.4.2.121 `conv_mix_bot`** `int ctl_t::conv_mix_bot`

Lower level for mixing (0=particle pressure, 1=surface).

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

**4.4.2.122 `conv_mix_top`** `int ctl_t::conv_mix_top`

Upper level for mixing (0=particle pressure, 1=EL).

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

**4.4.2.123 `bound_mass`** `double ctl_t::bound_mass`

Boundary conditions mass per particle [kg].

Definition at line 1061 of file [libtrac.h](#).

**4.4.2.124 `bound_mass_trend`** `double ctl_t::bound_mass_trend`

Boundary conditions mass per particle trend [kg/s].

Definition at line 1064 of file [libtrac.h](#).

**4.4.2.125 `bound_vmr`** `double ctl_t::bound_vmr`

Boundary conditions volume mixing ratio [ppv].

Definition at line 1067 of file [libtrac.h](#).

**4.4.2.126 bound\_vmr\_trend** `double ctl_t::bound_vmr_trend`

Boundary conditions volume mixing ratio trend [ppv/s].

Definition at line [1070](#) of file [libtrac.h](#).

**4.4.2.127 bound\_lat0** `double ctl_t::bound_lat0`

Boundary conditions minimum longitude [deg].

Definition at line [1073](#) of file [libtrac.h](#).

**4.4.2.128 bound\_lat1** `double ctl_t::bound_lat1`

Boundary conditions maximum longitude [deg].

Definition at line [1076](#) of file [libtrac.h](#).

**4.4.2.129 bound\_p0** `double ctl_t::bound_p0`

Boundary conditions bottom pressure [hPa].

Definition at line [1079](#) of file [libtrac.h](#).

**4.4.2.130 bound\_p1** `double ctl_t::bound_p1`

Boundary conditions top pressure [hPa].

Definition at line [1082](#) of file [libtrac.h](#).

**4.4.2.131 bound\_dps** `double ctl_t::bound_dps`

Boundary conditions delta to surface pressure [hPa].

Definition at line [1085](#) of file [libtrac.h](#).

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

Species.

Definition at line 1088 of file [libtrac.h](#).

**4.4.2.133 molmass** `double ctl_t::molmass`

Molar mass [g/mol].

Definition at line 1091 of file [libtrac.h](#).

**4.4.2.134 tdec\_trop** `double ctl_t::tdec_trop`

Life time of particles (troposphere) [s].

Definition at line 1094 of file [libtrac.h](#).

**4.4.2.135 tdec\_strat** `double ctl_t::tdec_strat`

Life time of particles (stratosphere) [s].

Definition at line 1097 of file [libtrac.h](#).

**4.4.2.136 clim\_oh\_filename** `char ctl_t::clim_oh_filename[LEN]`

Filename of OH climatology.

Definition at line 1100 of file [libtrac.h](#).

**4.4.2.137 clim\_h2o2\_filename** `char ctl_t::clim_h2o2_filename[LEN]`

Filename of H2O2 climatology.

Definition at line 1103 of file [libtrac.h](#).

**4.4.2.138 oh\_chem\_reaction** `int ctl_t::oh_chem_reaction`

Reaction type for OH chemistry (0=none, 2=bimolecular, 3=termolecular).

Definition at line [1106](#) of file [libtrac.h](#).

**4.4.2.139 oh\_chem** `double ctl_t::oh_chem[4]`

Coefficients for OH reaction rate (A, E/R or k0, n, kinf, m).

Definition at line [1109](#) of file [libtrac.h](#).

**4.4.2.140 oh\_chem\_beta** `double ctl_t::oh_chem_beta`

Beta parameter for diurnal variability of OH.

Definition at line [1112](#) of file [libtrac.h](#).

**4.4.2.141 h2o2\_chem\_cc** `double ctl_t::h2o2_chem_cc`

Cloud cover parameter for H2O2 chemistry.

Definition at line [1115](#) of file [libtrac.h](#).

**4.4.2.142 h2o2\_chem\_reaction** `int ctl_t::h2o2_chem_reaction`

Reaction type for H2O2 chemistry (0=none, 1=SO2).

Definition at line [1118](#) of file [libtrac.h](#).

**4.4.2.143 dry\_depo** `double ctl_t::dry_depo[1]`

Coefficients for dry deposition (v).

Definition at line [1121](#) of file [libtrac.h](#).



**4.4.2.144 `wet_depo_pre`** `double ctl_t::wet_depo_pre[2]`

Coefficients for precipitation calculation.

Definition at line 1124 of file [libtrac.h](#).

**4.4.2.145 `wet_depo_bc_a`** `double ctl_t::wet_depo_bc_a`

Coefficient A for wet deposition below cloud (exponential form).

Definition at line 1127 of file [libtrac.h](#).

**4.4.2.146 `wet_depo_bc_b`** `double ctl_t::wet_depo_bc_b`

Coefficient B for wet deposition below cloud (exponential form).

Definition at line 1130 of file [libtrac.h](#).

**4.4.2.147 `wet_depo_ic_a`** `double ctl_t::wet_depo_ic_a`

Coefficient A for wet deposition in cloud (exponential form).

Definition at line 1133 of file [libtrac.h](#).

**4.4.2.148 `wet_depo_ic_b`** `double ctl_t::wet_depo_ic_b`

Coefficient B for wet deposition in cloud (exponential form).

Definition at line 1136 of file [libtrac.h](#).

**4.4.2.149 `wet_depo_ic_h`** `double ctl_t::wet_depo_ic_h[3]`

Coefficients for wet deposition in cloud (Henry's law: Hb, Cb, pH).

Definition at line 1139 of file [libtrac.h](#).

**4.4.2.150 wet\_depo\_bc\_h** `double ctl_t::wet_depo_bc_h[2]`

Coefficients for wet deposition below cloud (Henry's law: Hb, Cb).

Definition at line [1142](#) of file [libtrac.h](#).

**4.4.2.151 wet\_depo\_ic\_ret\_ratio** `double ctl_t::wet_depo_ic_ret_ratio`

Coefficients for wet deposition in cloud: retention ratio.

Definition at line [1145](#) of file [libtrac.h](#).

**4.4.2.152 wet\_depo\_bc\_ret\_ratio** `double ctl_t::wet_depo_bc_ret_ratio`

Coefficients for wet deposition below cloud: retention ratio.

Definition at line [1148](#) of file [libtrac.h](#).

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

H2O volume mixing ratio for PSC analysis.

Definition at line [1151](#) of file [libtrac.h](#).

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

HNO3 volume mixing ratio for PSC analysis.

Definition at line [1154](#) of file [libtrac.h](#).

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

Basename of atmospheric data files.

Definition at line [1157](#) of file [libtrac.h](#).

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

Gnuplot file for atmospheric data.

Definition at line 1160 of file `libtrac.h`.

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

Time step for atmospheric data output [s].

Definition at line 1163 of file `libtrac.h`.

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

Time filter for atmospheric data output (0=none, 1=missval, 2=remove).

Definition at line 1166 of file `libtrac.h`.

**4.4.2.159 atm\_stride** `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line 1169 of file `libtrac.h`.

**4.4.2.160 atm\_type** `int ctl_t::atm_type`

Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF, 3=CLaMS).

Definition at line 1172 of file `libtrac.h`.

**4.4.2.161 csi\_basename** `char ctl_t::csi_basename[LEN]`

Basename of CSI data files.

Definition at line 1175 of file `libtrac.h`.

**4.4.2.162** `csi_dt_out` `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line [1178](#) of file [libtrac.h](#).

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

Observation data file for CSI analysis.

Definition at line [1181](#) of file [libtrac.h](#).

**4.4.2.164** `csi_obsmin` `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line [1184](#) of file [libtrac.h](#).

**4.4.2.165** `csi_modmin` `double ctl_t::csi_modmin`

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

Definition at line [1187](#) of file [libtrac.h](#).

**4.4.2.166** `csi_nz` `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line [1190](#) of file [libtrac.h](#).

**4.4.2.167** `csi_z0` `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

Definition at line [1193](#) of file [libtrac.h](#).

**4.4.2.168** `csi_z1` `double` `ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

Definition at line 1196 of file [libtrac.h](#).

**4.4.2.169** `csi_nx` `int` `ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

Definition at line 1199 of file [libtrac.h](#).

**4.4.2.170** `csi_lon0` `double` `ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 1202 of file [libtrac.h](#).

**4.4.2.171** `csi_lon1` `double` `ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 1205 of file [libtrac.h](#).

**4.4.2.172** `csi_ny` `int` `ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 1208 of file [libtrac.h](#).

**4.4.2.173** `csi_lat0` `double` `ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 1211 of file [libtrac.h](#).

**4.4.2.174** **csi\_lat1** `double ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line [1214](#) of file [libtrac.h](#).

**4.4.2.175** **ens\_basename** `char ctl_t::ens_basename[LEN]`

Basename of ensemble data file.

Definition at line [1217](#) of file [libtrac.h](#).

**4.4.2.176** **ens\_dt\_out** `double ctl_t::ens_dt_out`

Time step for ensemble output [s].

Definition at line [1220](#) of file [libtrac.h](#).

**4.4.2.177** **grid\_basename** `char ctl_t::grid_basename[LEN]`

Basename of grid data files.

Definition at line [1223](#) of file [libtrac.h](#).

**4.4.2.178** **grid\_gpfile** `char ctl_t::grid_gpfile[LEN]`

Gnuplot file for gridded data.

Definition at line [1226](#) of file [libtrac.h](#).

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

Time step for gridded data output [s].

Definition at line [1229](#) of file [libtrac.h](#).

**4.4.2.180 `grid_sparse`** `int ctl_t::grid_sparse`

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

Definition at line 1232 of file [libtrac.h](#).

**4.4.2.181 `grid_nz`** `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line 1235 of file [libtrac.h](#).

**4.4.2.182 `grid_z0`** `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line 1238 of file [libtrac.h](#).

**4.4.2.183 `grid_z1`** `double ctl_t::grid_z1`

Upper altitude of gridded data [km].

Definition at line 1241 of file [libtrac.h](#).

**4.4.2.184 `grid_nx`** `int ctl_t::grid_nx`

Number of longitudes of gridded data.

Definition at line 1244 of file [libtrac.h](#).

**4.4.2.185 `grid_lon0`** `double ctl_t::grid_lon0`

Lower longitude of gridded data [deg].

Definition at line 1247 of file [libtrac.h](#).

**4.4.2.186 grid\_lon1** `double ctl_t::grid_lon1`

Upper longitude of gridded data [deg].

Definition at line [1250](#) of file [libtrac.h](#).

**4.4.2.187 grid\_ny** `int ctl_t::grid_ny`

Number of latitudes of gridded data.

Definition at line [1253](#) of file [libtrac.h](#).

**4.4.2.188 grid\_lat0** `double ctl_t::grid_lat0`

Lower latitude of gridded data [deg].

Definition at line [1256](#) of file [libtrac.h](#).

**4.4.2.189 grid\_lat1** `double ctl_t::grid_lat1`

Upper latitude of gridded data [deg].

Definition at line [1259](#) of file [libtrac.h](#).

**4.4.2.190 grid\_type** `int ctl_t::grid_type`

Type of grid data files (0=ASCII, 1=netCDF).

Definition at line [1262](#) of file [libtrac.h](#).

**4.4.2.191 prof\_basename** `char ctl_t::prof_basename[LEN]`

Basename for profile output file.

Definition at line [1265](#) of file [libtrac.h](#).



**4.4.2.192 `prof_obsfile`** `char ctl_t::prof_obsfile[LEN]`

Observation data file for profile output.

Definition at line 1268 of file `libtrac.h`.

**4.4.2.193 `prof_nz`** `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line 1271 of file `libtrac.h`.

**4.4.2.194 `prof_z0`** `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line 1274 of file `libtrac.h`.

**4.4.2.195 `prof_z1`** `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line 1277 of file `libtrac.h`.

**4.4.2.196 `prof_nx`** `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line 1280 of file `libtrac.h`.

**4.4.2.197 `prof_lon0`** `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line 1283 of file `libtrac.h`.

**4.4.2.198 prof\_lon1** `double ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line [1286](#) of file [libtrac.h](#).

**4.4.2.199 prof\_ny** `int ctl_t::prof_ny`

Number of latitudes of gridded profile data.

Definition at line [1289](#) of file [libtrac.h](#).

**4.4.2.200 prof\_lat0** `double ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

Definition at line [1292](#) of file [libtrac.h](#).

**4.4.2.201 prof\_lat1** `double ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

Definition at line [1295](#) of file [libtrac.h](#).

**4.4.2.202 sample\_basename** `char ctl_t::sample_basename[LEN]`

Basename of sample data file.

Definition at line [1298](#) of file [libtrac.h](#).

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

Observation data file for sample output.

Definition at line [1301](#) of file [libtrac.h](#).

**4.4.2.204 `sample_dx`** `double ctl_t::sample_dx`

Horizontal radius for sample output [km].

Definition at line 1304 of file [libtrac.h](#).

**4.4.2.205 `sample_dz`** `double ctl_t::sample_dz`

Layer depth for sample output [km].

Definition at line 1307 of file [libtrac.h](#).

**4.4.2.206 `stat_basename`** `char ctl_t::stat_basename[LEN]`

Basename of station data file.

Definition at line 1310 of file [libtrac.h](#).

**4.4.2.207 `stat_lon`** `double ctl_t::stat_lon`

Longitude of station [deg].

Definition at line 1313 of file [libtrac.h](#).

**4.4.2.208 `stat_lat`** `double ctl_t::stat_lat`

Latitude of station [deg].

Definition at line 1316 of file [libtrac.h](#).

**4.4.2.209 `stat_r`** `double ctl_t::stat_r`

Search radius around station [km].

Definition at line 1319 of file [libtrac.h](#).

**4.4.2.210 stat\_t0** `double ctl_t::stat_t0`

Start time for station output [s].

Definition at line [1322](#) of file [libtrac.h](#).

**4.4.2.211 stat\_t1** `double ctl_t::stat_t1`

Stop time for station output [s].

Definition at line [1325](#) of file [libtrac.h](#).

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

- [libtrac.h](#)

## 4.5 met\_t Struct Reference

Meteo 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 [pbl](#) [EX][EY]  
*Boundary layer pressure [hPa].*
- 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 [pct](#) [EX][EY]  
*Cloud top pressure [hPa].*
- float [pcb](#) [EX][EY]  
*Cloud bottom 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 [cin](#) [EX][EY]  
*Convective inhibition [J/kg].*
- float [z](#) [EX][EY][EP]  
*Geopotential height [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].*
- float [patp](#) [EX][EY][EP]  
*Pressure field in pressure levels [hPa].*
- float [zeta](#) [EX][EY][EP]  
*Zeta [K].*
- float [zeta\\_dot](#) [EX][EY][EP]  
*Vertical velocity [K/s].*

#### 4.5.1 Detailed Description

Meteo data.

Definition at line [1459](#) of file [libtrac.h](#).

#### 4.5.2 Field Documentation

##### 4.5.2.1 **time** `double met_t::time`

Time [s].

Definition at line [1462](#) of file [libtrac.h](#).

##### 4.5.2.2 **nx** `int met_t::nx`

Number of longitudes.

Definition at line [1465](#) of file [libtrac.h](#).

##### 4.5.2.3 **ny** `int met_t::ny`

Number of latitudes.

Definition at line [1468](#) of file [libtrac.h](#).

##### 4.5.2.4 **np** `int met_t::np`

Number of pressure levels.

Definition at line [1471](#) of file [libtrac.h](#).

##### 4.5.2.5 **lon** `double met_t::lon[EX]`

Longitude [deg].

Definition at line [1474](#) of file [libtrac.h](#).

**4.5.2.6 lat** double met\_t::lat [\[EY\]](#)

Latitude [deg].

Definition at line [1477](#) of file [libtrac.h](#).

**4.5.2.7 p** double met\_t::p [\[EP\]](#)

Pressure [hPa].

Definition at line [1480](#) of file [libtrac.h](#).

**4.5.2.8 ps** float met\_t::ps [\[EX\]](#) [\[EY\]](#)

Surface pressure [hPa].

Definition at line [1483](#) of file [libtrac.h](#).

**4.5.2.9 ts** float met\_t::ts [\[EX\]](#) [\[EY\]](#)

Surface temperature [K].

Definition at line [1486](#) of file [libtrac.h](#).

**4.5.2.10 zs** float met\_t::zs [\[EX\]](#) [\[EY\]](#)

Surface geopotential height [km].

Definition at line [1489](#) of file [libtrac.h](#).

**4.5.2.11 us** float met\_t::us [\[EX\]](#) [\[EY\]](#)

Surface zonal wind [m/s].

Definition at line [1492](#) of file [libtrac.h](#).

**4.5.2.12 vs** `float met_t::vs`[\[EX\]](#) [\[EY\]](#)

Surface meridional wind [m/s].

Definition at line [1495](#) of file [libtrac.h](#).

**4.5.2.13 pbl** `float met_t::pbl`[\[EX\]](#) [\[EY\]](#)

Boundary layer pressure [hPa].

Definition at line [1498](#) of file [libtrac.h](#).

**4.5.2.14 pt** `float met_t::pt`[\[EX\]](#) [\[EY\]](#)

Tropopause pressure [hPa].

Definition at line [1501](#) of file [libtrac.h](#).

**4.5.2.15 tt** `float met_t::tt`[\[EX\]](#) [\[EY\]](#)

Tropopause temperature [K].

Definition at line [1504](#) of file [libtrac.h](#).

**4.5.2.16 zt** `float met_t::zt`[\[EX\]](#) [\[EY\]](#)

Tropopause geopotential height [km].

Definition at line [1507](#) of file [libtrac.h](#).

**4.5.2.17 h2ot** `float met_t::h2ot`[\[EX\]](#) [\[EY\]](#)

Tropopause water vapor vmr [ppv].

Definition at line [1510](#) of file [libtrac.h](#).



**4.5.2.18** **pct** float met\_t::pct [\[EX\]](#) [\[EY\]](#)

Cloud top pressure [hPa].

Definition at line [1513](#) of file [libtrac.h](#).

**4.5.2.19** **pcb** float met\_t::pcb [\[EX\]](#) [\[EY\]](#)

Cloud bottom pressure [hPa].

Definition at line [1516](#) of file [libtrac.h](#).

**4.5.2.20** **cl** float met\_t::cl [\[EX\]](#) [\[EY\]](#)

Total column cloud water [ $\text{kg/m}^2$ ].

Definition at line [1519](#) of file [libtrac.h](#).

**4.5.2.21** **plcl** float met\_t::plcl [\[EX\]](#) [\[EY\]](#)

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

Definition at line [1522](#) of file [libtrac.h](#).

**4.5.2.22** **plfc** float met\_t::plfc [\[EX\]](#) [\[EY\]](#)

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

Definition at line [1525](#) of file [libtrac.h](#).

**4.5.2.23** **pel** float met\_t::pel [\[EX\]](#) [\[EY\]](#)

Pressure at equilibrium level [hPa].

Definition at line [1528](#) of file [libtrac.h](#).

**4.5.2.24 cape** float met\_t::cape[EX][EY]

Convective available potential energy [J/kg].

Definition at line 1531 of file libtrac.h.

**4.5.2.25 cin** float met\_t::cin[EX][EY]

Convective inhibition [J/kg].

Definition at line 1534 of file libtrac.h.

**4.5.2.26 z** float met\_t::z[EX][EY][EP]

Geopotential height [km].

Definition at line 1537 of file libtrac.h.

**4.5.2.27 t** float met\_t::t[EX][EY][EP]

Temperature [K].

Definition at line 1540 of file libtrac.h.

**4.5.2.28 u** float met\_t::u[EX][EY][EP]

Zonal wind [m/s].

Definition at line 1543 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 1546 of file libtrac.h.

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

Vertical velocity [hPa/s].

Definition at line 1549 of file libtrac.h.

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

Potential vorticity [PVU].

Definition at line 1552 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 1555 of file libtrac.h.

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

Ozone volume mixing ratio [1].

Definition at line 1558 of file libtrac.h.

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

Cloud liquid water content [kg/kg].

Definition at line 1561 of file libtrac.h.

**4.5.2.35 iwc** float met\_t::iwc[EX][EY][EP]

Cloud ice water content [kg/kg].

Definition at line 1564 of file libtrac.h.

**4.5.2.36** **pl** float met\_t::pl[EX][EY][EP]

Pressure on model levels [hPa].

Definition at line 1567 of file libtrac.h.

**4.5.2.37** **patp** float met\_t::patp[EX][EY][EP]

Pressure field in pressure levels [hPa].

Definition at line 1570 of file libtrac.h.

**4.5.2.38** **zeta** float met\_t::zeta[EX][EY][EP]

Zeta [K].

Definition at line 1573 of file libtrac.h.

**4.5.2.39** **zeta\_dot** float met\_t::zeta\_dot[EX][EY][EP]

Vertical velocity [K/s].

Definition at line 1576 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

Convert file format of air parcel data files.

```
#include "libtrac.h"
```

#### 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](#).

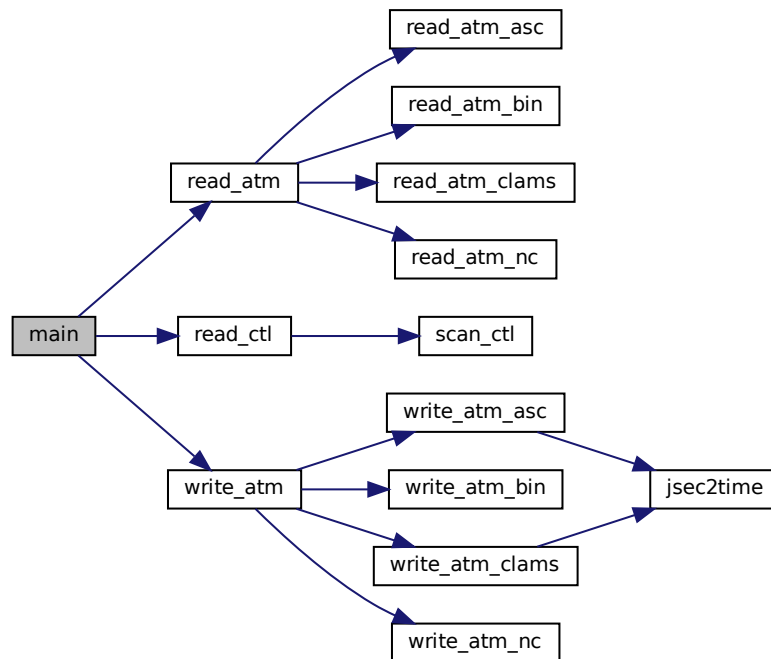
### 5.1.2 Function Documentation

**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) 2013-2022 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

Calculate transport deviations of trajectories.

```
#include "libtrac.h"
```

### 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, zscore;
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  zscore = scan_ctl(argv[1], argc, argv, "DIST_ZSCORE", -1, "-999", NULL);
00099
00100  /* Write info... */
00101  LOG(1, "Write transport deviations: %s", argv[2]);
00102
00103  /* Create output file... */
00104  if (!(out = fopen(argv[2], "w")))
00105      ERRMSG("Cannot create file!");
00106
00107  /* Write header... */
00108  fprintf(out,
00109      "# $1 = time [s]\n"
00110      "# $2 = time difference [s]\n"
00111      "# $3 = absolute horizontal distance (%s) [km]\n"
00112      "# $4 = relative horizontal distance (%s) [%%]\n"
00113      "# $5 = absolute vertical distance (%s) [km]\n"
00114      "# $6 = relative vertical distance (%s) [%%]\n",
00115      argv[3], argv[3], argv[3], argv[3]);
00116  for (iq = 0; iq < ctl.nq; iq++)
00117      fprintf(out,
00118          "# %d = %s absolute difference (%s) [%s]\n"
00119          "# %d = %s relative difference (%s) [%%]\n",
00120          7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00121          8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00122  fprintf(out, "# %d = number of particles\n\n", 7 + 2 * ctl.nq);
00123
00124  /* Loop over file pairs... */
00125  for (f = 4; f < argc; f += 2) {
00126
00127      /* Read atmospheric data... */
00128      if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00129          continue;
00130
00131      /* Check if structs match... */
00132      if (atm1->np != atm2->np)
00133          ERRMSG("Different numbers of particles!");
00134
00135      /* Get time from filename... */

```



```

00136     size_t len = strlen(argv[f]);
00137     sprintf(tstr, "%.4s", &argv[f][len - 20]);
00138     year = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][len - 15]);
00140     mon = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][len - 12]);
00142     day = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][len - 9]);
00144     hour = atoi(tstr);
00145     sprintf(tstr, "%.2s", &argv[f][len - 6]);
00146     min = atoi(tstr);
00147     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00148
00149     /* Check time... */
00150     if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00151         || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00152         ERRMSG("Cannot read time from filename!");
00153
00154     /* Save initial time... */
00155     if (!init) {
00156         init = 1;
00157         t0 = t;
00158     }
00159
00160     /* Init... */
00161     np = 0;
00162     for (ip = 0; ip < atm1->np; ip++) {
00163         ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00164         for (iq = 0; iq < ctl.nq; iq++)
00165             aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00166     }
00167
00168     /* Loop over air parcels... */
00169     for (ip = 0; ip < atm1->np; ip++) {
00170
00171         /* Check air parcel index... */
00172         if (ctl.qnt_idx > 0
00173             && (atm1->q[ctl.qnt_idx][ip] != atm2->q[ctl.qnt_idx][ip]))
00174             ERRMSG("Air parcel index does not match!");
00175
00176         /* Check ensemble index... */
00177         if (ctl.qnt_ens > 0
00178             && (atm1->q[ctl.qnt_ens][ip] != ens
00179                 || atm2->q[ctl.qnt_ens][ip] != ens))
00180             continue;
00181
00182         /* Check time... */
00183         if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00184             continue;
00185
00186         /* Check spatial range... */
00187         if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00188             || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00189             || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00190             continue;
00191         if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00192             || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00193             || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00194             continue;
00195
00196         /* Convert coordinates... */
00197         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00198         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00199         z1 = Z(atm1->p[ip]);
00200         z2 = Z(atm2->p[ip]);
00201
00202         /* Calculate absolute transport deviations... */
00203         ahtd[np] = DIST(x1, x2);
00204         avtd[np] = z1 - z2;
00205         for (iq = 0; iq < ctl.nq; iq++)
00206             aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00207
00208         /* Calculate relative transport deviations... */
00209         if (f > 4) {
00210
00211             /* Get trajectory lengths... */
00212             geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00213             lh1[ip] += DIST(x0, x1);
00214             lv1[ip] += fabs(z1_old[ip] - z1);
00215
00216             geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00217             lh2[ip] += DIST(x0, x2);
00218             lv2[ip] += fabs(z2_old[ip] - z2);
00219
00220             /* Get relative transport deviations... */
00221             if (lh1[ip] + lh2[ip] > 0)
00222                 rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);

```

```

00223         if (lv1[ip] + lv2[ip] > 0)
00224             rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00225     }
00226
00227     /* Get relative transport deviations... */
00228     for (iq = 0; iq < ctl.nq; iq++)
00229         rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00230         / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00231
00232     /* Save positions of air parcels... */
00233     lon1_old[ip] = atm1->lon[ip];
00234     lat1_old[ip] = atm1->lat[ip];
00235     z1_old[ip] = z1;
00236
00237     lon2_old[ip] = atm2->lon[ip];
00238     lat2_old[ip] = atm2->lat[ip];
00239     z2_old[ip] = z2;
00240
00241     /* Increment air parcel counter... */
00242     np++;
00243 }
00244
00245 /* Filter data... */
00246 if (zscore > 0 && np > 1) {
00247
00248     /* Get means and standard deviations of transport deviations... */
00249     size_t n = (size_t) np;
00250     double muh = gsl_stats_mean(ahtd, 1, n);
00251     double muv = gsl_stats_mean(avtd, 1, n);
00252     double sigh = gsl_stats_sd(ahtd, 1, n);
00253     double sigv = gsl_stats_sd(avtd, 1, n);
00254
00255     /* Filter data... */
00256     np = 0;
00257     for (size_t i = 0; i < n; i++)
00258         if (fabs((ahtd[i] - muh) / sigh) < zscore
00259             && fabs((avtd[i] - muv) / sigv) < zscore) {
00260             ahtd[np] = ahtd[i];
00261             rhtd[np] = rhtd[i];
00262             avtd[np] = avtd[i];
00263             rvtd[np] = rvtd[i];
00264             for (iq = 0; iq < ctl.nq; iq++) {
00265                 aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
00266                 rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
00267             }
00268             np++;
00269         }
00270 }
00271
00272 /* Get statistics... */
00273 if (strcasecmp(argv[3], "mean") == 0) {
00274     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00275     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00276     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00277     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00278     for (iq = 0; iq < ctl.nq; iq++) {
00279         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00280         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00281     }
00282 } else if (strcasecmp(argv[3], "stddev") == 0) {
00283     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00284     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00285     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00286     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00287     for (iq = 0; iq < ctl.nq; iq++) {
00288         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00289         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00290     }
00291 } else if (strcasecmp(argv[3], "min") == 0) {
00292     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00293     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00294     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00295     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00296     for (iq = 0; iq < ctl.nq; iq++) {
00297         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00298         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00299     }
00300 } else if (strcasecmp(argv[3], "max") == 0) {
00301     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00302     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00303     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00304     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00305     for (iq = 0; iq < ctl.nq; iq++) {
00306         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00307         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00308     }
00309 } else if (strcasecmp(argv[3], "skew") == 0) {

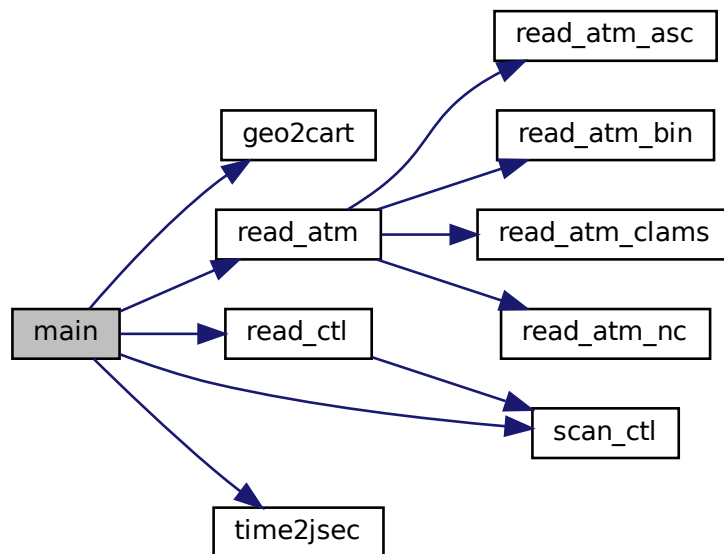
```

```

00310     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00311     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00312     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00313     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00314     for (iq = 0; iq < ctl.nq; iq++) {
00315         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00316         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00317     }
00318 } else if (strcasecmp(argv[3], "kurt") == 0) {
00319     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00320     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00321     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00322     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00323     for (iq = 0; iq < ctl.nq; iq++) {
00324         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00325         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00326     }
00327 } else if (strcasecmp(argv[3], "absdev") == 0) {
00328     ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
00329     rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
00330     avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
00331     rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
00332     for (iq = 0; iq < ctl.nq; iq++) {
00333         aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
00334         rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
00335     }
00336 } else if (strcasecmp(argv[3], "median") == 0) {
00337     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00338     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00339     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00340     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00341     for (iq = 0; iq < ctl.nq; iq++) {
00342         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00343         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00344     }
00345 } else if (strcasecmp(argv[3], "mad") == 0) {
00346     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00347     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00348     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00349     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00350     for (iq = 0; iq < ctl.nq; iq++) {
00351         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00352         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00353     }
00354 } else
00355     ERRMSG("Unknown parameter!");
00356
00357 /* Write output... */
00358 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00359         ahtdm, rhtdm, avtdm, rvtdm);
00360 for (iq = 0; iq < ctl.nq; iq++) {
00361     fprintf(out, " ");
00362     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00363     fprintf(out, " ");
00364     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00365 }
00366 fprintf(out, " %d\n", np);
00367 }
00368
00369 /* Close file... */
00370 fclose(out);
00371
00372 /* Free... */
00373 free(atm1);
00374 free(atm2);
00375 free(lon1_old);
00376 free(lat1_old);
00377 free(z1_old);
00378 free(lh1);
00379 free(lv1);
00380 free(lon2_old);
00381 free(lat2_old);
00382 free(z2_old);
00383 free(lh2);
00384 free(lv2);
00385 free(ahtd);
00386 free(avtd);
00387 free(aqtd);
00388 free(rhtd);
00389 free(rvtd);
00390 free(rqtd);
00391 free(work);
00392
00393 return EXIT_SUCCESS;
00394 }

```

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) 2013-2022 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm1, *atm2;
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, zscore;
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     zscore = scan_ctl(argv[1], argc, argv, "DIST_ZSCORE", -1, "-999", NULL);
00099
00100     /* Write info... */
00101     LOG(1, "Write transport deviations: %s", argv[2]);
00102
00103     /* Create output file... */
00104     if (!(out = fopen(argv[2], "w")))
00105         ERRMSG("Cannot create file!");
00106
00107     /* Write header... */
00108     fprintf(out,
00109         "# $1 = time [s]\n"
00110         "# $2 = time difference [s]\n"
00111         "# $3 = absolute horizontal distance (%s) [km]\n"
00112         "# $4 = relative horizontal distance (%s) [%%]\n"
00113         "# $5 = absolute vertical distance (%s) [km]\n"
00114         "# $6 = relative vertical distance (%s) [%%]\n",
00115         argv[3], argv[3], argv[3], argv[3]);
00116     for (iq = 0; iq < ctl.nq; iq++)
00117         fprintf(out,
00118             "# $%d = %s absolute difference (%s) [%s]\n"
00119             "# $%d = %s relative difference (%s) [%%]\n",
00120             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00121             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00122     fprintf(out, "# $%d = number of particles\n\n", 7 + 2 * ctl.nq);
00123
00124     /* Loop over file pairs... */
00125     for (f = 4; f < argc; f += 2) {
00126
00127         /* Read atmospheric data... */
00128         if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00129             continue;
00130
00131         /* Check if structs match... */
00132         if (atml->np != atm2->np)
00133             ERRMSG("Different numbers of particles!");
00134

```

```

00135     /* Get time from filename... */
00136     size_t len = strlen(argv[f]);
00137     sprintf(tstr, "%.4s", &argv[f][len - 20]);
00138     year = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][len - 15]);
00140     mon = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][len - 12]);
00142     day = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][len - 9]);
00144     hour = atoi(tstr);
00145     sprintf(tstr, "%.2s", &argv[f][len - 6]);
00146     min = atoi(tstr);
00147     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00148
00149     /* Check time... */
00150     if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00151         || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00152         ERRMSG("Cannot read time from filename!");
00153
00154     /* Save initial time... */
00155     if (!init) {
00156         init = 1;
00157         t0 = t;
00158     }
00159
00160     /* Init... */
00161     np = 0;
00162     for (ip = 0; ip < atm1->np; ip++) {
00163         ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00164         for (iq = 0; iq < ctl.nq; iq++)
00165             aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00166     }
00167
00168     /* Loop over air parcels... */
00169     for (ip = 0; ip < atm1->np; ip++) {
00170
00171         /* Check air parcel index... */
00172         if (ctl.qnt_idx > 0
00173             && (atm1->q[ctl.qnt_idx][ip] != atm2->q[ctl.qnt_idx][ip]))
00174             ERRMSG("Air parcel index does not match!");
00175
00176         /* Check ensemble index... */
00177         if (ctl.qnt_ens > 0
00178             && (atm1->q[ctl.qnt_ens][ip] != ens
00179                 || atm2->q[ctl.qnt_ens][ip] != ens))
00180             continue;
00181
00182         /* Check time... */
00183         if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00184             continue;
00185
00186         /* Check spatial range... */
00187         if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00188             || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00189             || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00190             continue;
00191         if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00192             || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00193             || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00194             continue;
00195
00196         /* Convert coordinates... */
00197         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00198         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00199         z1 = Z(atm1->p[ip]);
00200         z2 = Z(atm2->p[ip]);
00201
00202         /* Calculate absolute transport deviations... */
00203         ahtd[np] = DIST(x1, x2);
00204         avtd[np] = z1 - z2;
00205         for (iq = 0; iq < ctl.nq; iq++)
00206             aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00207
00208         /* Calculate relative transport deviations... */
00209         if (f > 4) {
00210
00211             /* Get trajectory lengths... */
00212             geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00213             lh1[ip] += DIST(x0, x1);
00214             lv1[ip] += fabs(z1_old[ip] - z1);
00215
00216             geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00217             lh2[ip] += DIST(x0, x2);
00218             lv2[ip] += fabs(z2_old[ip] - z2);
00219
00220             /* Get relative transport deviations... */
00221             if (lh1[ip] + lh2[ip] > 0)

```

```

00222         rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00223         if (lv1[ip] + lv2[ip] > 0)
00224             rvtb[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00225     }
00226
00227     /* Get relative transport deviations... */
00228     for (iq = 0; iq < ctl.nq; iq++)
00229         rqtq[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00230             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00231
00232     /* Save positions of air parcels... */
00233     lon1_old[ip] = atm1->lon[ip];
00234     lat1_old[ip] = atm1->lat[ip];
00235     z1_old[ip] = z1;
00236
00237     lon2_old[ip] = atm2->lon[ip];
00238     lat2_old[ip] = atm2->lat[ip];
00239     z2_old[ip] = z2;
00240
00241     /* Increment air parcel counter... */
00242     np++;
00243 }
00244
00245 /* Filter data... */
00246 if (zscore > 0 && np > 1) {
00247
00248     /* Get means and standard deviations of transport deviations... */
00249     size_t n = (size_t) np;
00250     double muh = gsl_stats_mean(ahtd, 1, n);
00251     double muv = gsl_stats_mean(avtd, 1, n);
00252     double sigh = gsl_stats_sd(ahtd, 1, n);
00253     double sigv = gsl_stats_sd(avtd, 1, n);
00254
00255     /* Filter data... */
00256     np = 0;
00257     for (size_t i = 0; i < n; i++)
00258         if (fabs((ahtd[i] - muh) / sigh) < zscore
00259             && fabs((avtd[i] - muv) / sigv) < zscore) {
00260             ahtd[np] = ahtd[i];
00261             rhtd[np] = rhtd[i];
00262             avtd[np] = avtd[i];
00263             rvtb[np] = rvtb[i];
00264             for (iq = 0; iq < ctl.nq; iq++) {
00265                 aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
00266                 rqtq[iq * NP + np] = rqtq[iq * NP + (int) i];
00267             }
00268             np++;
00269         }
00270 }
00271
00272 /* Get statistics... */
00273 if (strcmp(argv[3], "mean") == 0) {
00274     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00275     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00276     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00277     rvtbm = gsl_stats_mean(rvtb, 1, (size_t) np);
00278     for (iq = 0; iq < ctl.nq; iq++) {
00279         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00280         rqtqm[iq] = gsl_stats_mean(&rqtq[iq * NP], 1, (size_t) np);
00281     }
00282 } else if (strcmp(argv[3], "stddev") == 0) {
00283     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00284     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00285     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00286     rvtbm = gsl_stats_sd(rvtb, 1, (size_t) np);
00287     for (iq = 0; iq < ctl.nq; iq++) {
00288         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00289         rqtqm[iq] = gsl_stats_sd(&rqtq[iq * NP], 1, (size_t) np);
00290     }
00291 } else if (strcmp(argv[3], "min") == 0) {
00292     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00293     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00294     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00295     rvtbm = gsl_stats_min(rvtb, 1, (size_t) np);
00296     for (iq = 0; iq < ctl.nq; iq++) {
00297         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00298         rqtqm[iq] = gsl_stats_min(&rqtq[iq * NP], 1, (size_t) np);
00299     }
00300 } else if (strcmp(argv[3], "max") == 0) {
00301     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00302     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00303     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00304     rvtbm = gsl_stats_max(rvtb, 1, (size_t) np);
00305     for (iq = 0; iq < ctl.nq; iq++) {
00306         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00307         rqtqm[iq] = gsl_stats_max(&rqtq[iq * NP], 1, (size_t) np);
00308     }

```

```

00309     } else if (strcasecmp(argv[3], "skew") == 0) {
00310         ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00311         rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00312         avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00313         rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00314         for (iq = 0; iq < ctl.nq; iq++) {
00315             aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00316             rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00317         }
00318     } else if (strcasecmp(argv[3], "kurt") == 0) {
00319         ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00320         rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00321         avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00322         rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00323         for (iq = 0; iq < ctl.nq; iq++) {
00324             aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00325             rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00326         }
00327     } else if (strcasecmp(argv[3], "absdev") == 0) {
00328         ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
00329         rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
00330         avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
00331         rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
00332         for (iq = 0; iq < ctl.nq; iq++) {
00333             aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
00334             rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
00335         }
00336     } else if (strcasecmp(argv[3], "median") == 0) {
00337         ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00338         rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00339         avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00340         rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00341         for (iq = 0; iq < ctl.nq; iq++) {
00342             aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00343             rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00344         }
00345     } else if (strcasecmp(argv[3], "mad") == 0) {
00346         ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00347         rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00348         avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00349         rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00350         for (iq = 0; iq < ctl.nq; iq++) {
00351             aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00352             rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00353         }
00354     } else
00355         ERRMSG("Unknown parameter!");
00356
00357     /* Write output... */
00358     fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00359             ahtdm, rhtdm, avtdm, rvtdm);
00360     for (iq = 0; iq < ctl.nq; iq++) {
00361         fprintf(out, " ");
00362         fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00363         fprintf(out, " ");
00364         fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00365     }
00366     fprintf(out, " %d\n", np);
00367 }
00368
00369 /* Close file... */
00370 fclose(out);
00371
00372 /* Free... */
00373 free(atm1);
00374 free(atm2);
00375 free(lon1_old);
00376 free(lat1_old);
00377 free(z1_old);
00378 free(lh1);
00379 free(lv1);
00380 free(lon2_old);
00381 free(lat2_old);
00382 free(z2_old);
00383 free(lh2);
00384 free(lv2);
00385 free(ahtd);
00386 free(avtd);
00387 free(aqtd);
00388 free(rhtd);
00389 free(rvtd);
00390 free(rqtd);
00391 free(work);
00392
00393 return EXIT_SUCCESS;
00394 }

```



## 5.5 atm\_init.c File Reference

Create atmospheric data file with initial air parcel positions.

```
#include "libtrac.h"
```

### 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, t, z,
00038            lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m, vmr, bellrad;
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 vmr = scan_ctl(argv[1], argc, argv, "INIT_VMR", -1, "0", NULL);
00076 bellrad = scan_ctl(argv[1], argc, argv, "INIT_BELLRAD", -1, "0", NULL);
00077
00078 /* Initialize random number generator... */
00079 gsl_rng_env_setup();
00080 rng = gsl_rng_alloc(gsl_rng_default);
00081
00082 /* Create grid... */
00083 for (t = t0; t <= t1; t += dt)
00084     for (z = z0; z <= z1; z += dz)
00085         for (lon = lon0; lon <= lon1; lon += dlon)
00086             for (lat = lat0; lat <= lat1; lat += dlat)
00087                 for (irep = 0; irep < rep; irep++) {
00088
00089                     /* Set position... */
00090                     atm->time[atm->np]
00091                         = (t + gsl_rng_gaussian_ziggurat(rng, st / 2.3548)
00092                            + ut * (gsl_rng_uniform(rng) - 0.5));
00093                     atm->p[atm->np]
00094                         = P(z + gsl_rng_gaussian_ziggurat(rng, sz / 2.3548)
00095                            + uz * (gsl_rng_uniform(rng) - 0.5));
00096                     atm->lon[atm->np]
00097                         = (lon + gsl_rng_gaussian_ziggurat(rng, slon / 2.3548)
00098                            + gsl_rng_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00099                            + ulon * (gsl_rng_uniform(rng) - 0.5));
00100                     do {
00101                         atm->lat[atm->np]
00102                             = (lat + gsl_rng_gaussian_ziggurat(rng, slat / 2.3548)
00103                                + gsl_rng_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00104                                + ulat * (gsl_rng_uniform(rng) - 0.5));
00105                     } while (even && gsl_rng_uniform(rng) >
00106                             fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00107
00108                     /* Apply cosine bell (Williamson et al., 1992)... */
00109                     if (bellrad > 0) {
00110                         double x0[3], x1[3];
00111                         geo2cart(0.0, 0.5 * (lon0 + lon1), 0.5 * (lat0 + lat1), x0);
00112                         geo2cart(0.0, atm->lon[atm->np], atm->lat[atm->np], x1);
00113                         double rad = RE * acos(DOTP(x0, x1) / NORM(x0) / NORM(x1));
00114                         if (rad > bellrad)
00115                             continue;
00116                         if (ctl.qnt_m >= 0)
00117                             atm->q[ctl.qnt_m][atm->np] =
00118                                 0.5 * (1. + cos(M_PI * rad / bellrad));
00119                         if (ctl.qnt_vmr >= 0)
00120                             atm->q[ctl.qnt_vmr][atm->np] =
00121                                 0.5 * (1. + cos(M_PI * rad / bellrad));
00122                     }
00123
00124                     /* Set particle counter... */
00125                     if ((++atm->np) > NP)
00126                         ERRMSG("Too many particles!");
00127                 }
00128
00129 /* Check number of air parcels... */
00130 if (atm->np <= 0)
00131     ERRMSG("Did not create any air parcels!");
00132
00133 /* Initialize mass... */
00134 if (ctl.qnt_m >= 0 && bellrad <= 0)
00135     for (ip = 0; ip < atm->np; ip++)
00136         atm->q[ctl.qnt_m][ip] = m / atm->np;
00137
00138 /* Initialize volume mixing ratio... */
00139 if (ctl.qnt_vmr >= 0 && bellrad <= 0)
00140     for (ip = 0; ip < atm->np; ip++)
00141         atm->q[ctl.qnt_vmr][ip] = vmr;
00142
00143 /* Initialize air parcel index... */
00144 if (ctl.qnt_idx >= 0)
00145     for (ip = 0; ip < atm->np; ip++)
00146         atm->q[ctl.qnt_idx][ip] = ip;
00147
00148 /* Save data... */
00149 write_atm(argv[2], &ctl, atm, 0);
00150
00151 /* Free... */
00152 gsl_rng_free(rng);
00153 free(atm);
00154

```

```
00155     return EXIT_SUCCESS;
00156 }
```

## 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) 2013-2022 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, t, z,
00038         lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m, vmr, bellrad;
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     vmr = scan_ctl(argv[1], argc, argv, "INIT_VMR", -1, "0", NULL);
00076     bellrad = scan_ctl(argv[1], argc, argv, "INIT_BELLRAD", -1, "0", NULL);
00077
00078     /* Initialize random number generator... */
00079     gsl_rng_env_setup();
00080     rng = gsl_rng_alloc(gsl_rng_default);
00081
00082     /* Create grid... */
00083     for (t = t0; t <= t1; t += dt)
00084         for (z = z0; z <= z1; z += dz)
```

```

00085     for (lon = lon0; lon <= lon1; lon += dlon)
00086         for (lat = lat0; lat <= lat1; lat += dlat)
00087             for (irep = 0; irep < rep; irep++) {
00088
00089                 /* Set position... */
00090                 atm->time[atm->np]
00091                     = (t + gsl_rng_gaussian_ziggurat(rng, st / 2.3548)
00092                        + ut * (gsl_rng_uniform(rng) - 0.5));
00093                 atm->p[atm->np]
00094                     = P(z + gsl_rng_gaussian_ziggurat(rng, sz / 2.3548)
00095                        + uz * (gsl_rng_uniform(rng) - 0.5));
00096                 atm->lon[atm->np]
00097                     = (lon + gsl_rng_gaussian_ziggurat(rng, slon / 2.3548)
00098                        + gsl_rng_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00099                        + ulon * (gsl_rng_uniform(rng) - 0.5));
00100                 do {
00101                     atm->lat[atm->np]
00102                         = (lat + gsl_rng_gaussian_ziggurat(rng, slat / 2.3548)
00103                            + gsl_rng_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00104                            + ulat * (gsl_rng_uniform(rng) - 0.5));
00105                     } while (even && gsl_rng_uniform(rng) >
00106                             fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00107
00108                 /* Apply cosine bell (Williamson et al., 1992)... */
00109                 if (bellrad > 0) {
00110                     double x0[3], x1[3];
00111                     geo2cart(0.0, 0.5 * (lon0 + lon1), 0.5 * (lat0 + lat1), x0);
00112                     geo2cart(0.0, atm->lon[atm->np], atm->lat[atm->np], x1);
00113                     double rad = RE * acos(DOTP(x0, x1) / NORM(x0) / NORM(x1));
00114                     if (rad > bellrad)
00115                         continue;
00116                     if (ctl.qnt_m >= 0)
00117                         atm->q[ctl.qnt_m][atm->np] =
00118                             0.5 * (1. + cos(M_PI * rad / bellrad));
00119                     if (ctl.qnt_vmr >= 0)
00120                         atm->q[ctl.qnt_vmr][atm->np] =
00121                             0.5 * (1. + cos(M_PI * rad / bellrad));
00122                 }
00123
00124                 /* Set particle counter... */
00125                 if ((++atm->np) > NP)
00126                     ERRMSG("Too many particles!");
00127             }
00128
00129     /* Check number of air parcels... */
00130     if (atm->np <= 0)
00131         ERRMSG("Did not create any air parcels!");
00132
00133     /* Initialize mass... */
00134     if (ctl.qnt_m >= 0 && bellrad <= 0)
00135         for (ip = 0; ip < atm->np; ip++)
00136             atm->q[ctl.qnt_m][ip] = m / atm->np;
00137
00138     /* Initialize volume mixing ratio... */
00139     if (ctl.qnt_vmr >= 0 && bellrad <= 0)
00140         for (ip = 0; ip < atm->np; ip++)
00141             atm->q[ctl.qnt_vmr][ip] = vmr;
00142
00143     /* Initialize air parcel index... */
00144     if (ctl.qnt_idx >= 0)
00145         for (ip = 0; ip < atm->np; ip++)
00146             atm->q[ctl.qnt_idx][ip] = ip;
00147
00148     /* Save data... */
00149     write_atm(argv[2], &ctl, atm, 0);
00150
00151     /* Free... */
00152     gsl_rng_free(rng);
00153     free(atm);
00154
00155     return EXIT_SUCCESS;
00156 }

```

## 5.7 atm\_select.c File Reference

Extract subsets of air parcels from atmospheric data files.

```
#include "libtrac.h"
```

## 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](#).

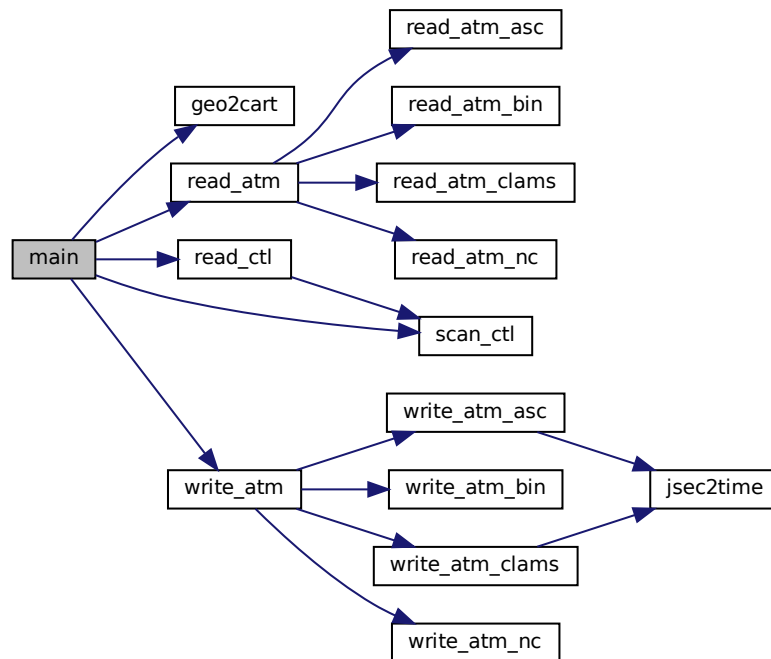
```
00029     {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm, *atm2;
00034
00035     double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
00036            x1[3];
00037
00038     int f, ip, idx0, idx1, ip0, ip1, iq, stride;
00039
00040     /* Allocate... */
00041     ALLOC(atm, 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     idx0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX0", -1, "-999", NULL);
00053     idx1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX1", -1, "-999", NULL);
00054     ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00055     ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00056     t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00057     t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00058     p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00059     p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00060     lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00061     lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00062     lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00063     lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00064     r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00065     r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00066     rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00067     rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00068
00069     /* Get Cartesian coordinates... */
00070     geo2cart(0, rlon, rlat, x0);
00071
00072     /* Loop over files... */
00073     for (f = 3; f < argc; f++) {
00074
00075         /* Read atmospheric data... */
00076         if (!read_atm(argv[f], &ctl, atm))
00077             continue;
00078
00079         /* Adjust range of air parcels... */
00080         if (ip0 < 0)
```

```

00081     ip0 = 0;
00082     ip0 = GSL_MIN(ip0, atm->np - 1);
00083     if (ip1 < 0)
00084         ip1 = atm->np - 1;
00085     ip1 = GSL_MIN(ip1, atm->np - 1);
00086     if (ip1 < ip0)
00087         ip1 = ip0;
00088
00089     /* Loop over air parcels... */
00090     for (ip = ip0; ip <= ip1; ip += stride) {
00091
00092         /* Check air parcel index... */
00093         if (ctl.qnt_idx >= 0 && idx0 >= 0 && idx1 >= 0)
00094             if (atm->q[ctl.qnt_idx][ip] < idx0 || atm->q[ctl.qnt_idx][ip] > idx1)
00095                 continue;
00096
00097         /* Check time... */
00098         if (t0 != t1)
00099             if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00100                 || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00101                 continue;
00102
00103         /* Check vertical distance... */
00104         if (p0 != p1)
00105             if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00106                 || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00107                 continue;
00108
00109         /* Check longitude... */
00110         if (lon0 != lon1)
00111             if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00112                 || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00113                 continue;
00114
00115         /* Check latitude... */
00116         if (lat0 != lat1)
00117             if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00118                 || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00119                 continue;
00120
00121         /* Check horizontal distace... */
00122         if (r0 != r1) {
00123             geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00124             r = DIST(x0, x1);
00125             if ((r1 > r0 && (r < r0 || r > r1))
00126                 || (r1 < r0 && (r < r0 && r > r1)))
00127                 continue;
00128         }
00129
00130         /* Copy data... */
00131         atm2->time[atm2->np] = atm->time[ip];
00132         atm2->p[atm2->np] = atm->p[ip];
00133         atm2->lon[atm2->np] = atm->lon[ip];
00134         atm2->lat[atm2->np] = atm->lat[ip];
00135         for (iq = 0; iq < ctl.nq; iq++)
00136             atm2->q[iq][atm2->np] = atm->q[iq][ip];
00137         if ((++atm2->np) > NP)
00138             ERRMSG("Too many air parcels!");
00139     }
00140 }
00141
00142 /* Close file... */
00143 write_atm(argv[2], &ctl, atm2, 0);
00144
00145 /* Free... */
00146 free(atm);
00147 free(atm2);
00148
00149 return EXIT_SUCCESS;
00150 }

```

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) 2013-2022 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, idx0, idx1, 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> <atml> [<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 idx0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX0", -1, "-999", NULL);
00053 idx1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX1", -1, "-999", NULL);
00054 ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00055 ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00056 t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00057 t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00058 p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00059 p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00060 lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00061 lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00062 lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00063 lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00064 r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00065 r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00066 rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00067 rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00068
00069 /* Get Cartesian coordinates... */
00070 geo2cart(0, rlon, rlat, x0);
00071
00072 /* Loop over files... */
00073 for (f = 3; f < argc; f++) {
00074
00075     /* Read atmospheric data... */
00076     if (!read_atm(argv[f], &ctl, atm))
00077         continue;
00078
00079     /* Adjust range of air parcels... */
00080     if (ip0 < 0)
00081         ip0 = 0;
00082     ip0 = GSL_MIN(ip0, atm->np - 1);
00083     if (ip1 < 0)
00084         ip1 = atm->np - 1;
00085     ip1 = GSL_MIN(ip1, atm->np - 1);
00086     if (ip1 < ip0)
00087         ip1 = ip0;
00088
00089     /* Loop over air parcels... */
00090     for (ip = ip0; ip <= ip1; ip += stride) {
00091
00092         /* Check air parcel index... */
00093         if (ctl.qnt_idx >= 0 && idx0 >= 0 && idx1 >= 0)
00094             if (atm->q[ctl.qnt_idx][ip] < idx0 || atm->q[ctl.qnt_idx][ip] > idx1)
00095                 continue;
00096
00097         /* Check time... */
00098         if (t0 != t1)
00099             if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00100                 || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00101                 continue;
00102
00103         /* Check vertical distance... */
00104         if (p0 != p1)
00105             if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00106                 || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00107                 continue;
00108
00109         /* Check longitude... */
00110         if (lon0 != lon1)
00111             if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00112                 || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00113                 continue;
00114
00115         /* Check latitude... */
00116         if (lat0 != lat1)
00117             if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00118                 || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00119                 continue;
00120
00121         /* Check horizontal distance... */
00122         if (r0 != r1) {
00123             geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00124             r = DIST(x0, x1);
00125             if ((r1 > r0 && (r < r0 || r > r1))
00126                 || (r1 < r0 && (r < r0 && r > r1)))
00127                 continue;
00128         }
00129
00130         /* Copy data... */
00131         atm2->time[atm2->np] = atm->time[ip];

```



```

00132     atm2->p[atm2->np] = atm->p[ip];
00133     atm2->lon[atm2->np] = atm->lon[ip];
00134     atm2->lat[atm2->np] = atm->lat[ip];
00135     for (iq = 0; iq < ctl.nq; iq++)
00136         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00137     if ((++atm2->np) > NP)
00138         ERRMSG("Too many air parcels!");
00139     }
00140 }
00141
00142 /* Close file... */
00143 write_atm(argv[2], &ctl, atm2, 0);
00144
00145 /* Free... */
00146 free(atm);
00147 free(atm2);
00148
00149 return EXIT_SUCCESS;
00150 }

```

## 5.9 atm\_split.c File Reference

Split air parcels into a larger number of parcels.

```
#include "libtrac.h"
```

### 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[EP], kz[EP], 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     LOG(1, "Read kernel function: %s", 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) >= EP)
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_rng_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_rng_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_rng_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00157         atm2->lat[atm2->np] = atm->lat[ip]
00158         + gsl_rng_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     /* Adjust air parcel index... */
00170     if (ctl.qnt_idx >= 0)
00171         atm2->q[ctl.qnt_idx][atm2->np] = atm2->np;
00172
00173     /* Increment particle counter... */
00174     if ((++atm2->np) > NP)
00175         ERRMSG("Too many air parcels!");
00176 }
00177
00178 /* Save data and close file... */
00179 write_atm(argv[3], &ctl, atm2, 0);
00180
00181 /* Free... */
00182 free(atm);
00183 free(atm2);
00184
00185 return EXIT_SUCCESS;
00186 }

```

## 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) 2013-2022 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[EP], kz[EP], 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         LOG(1, "Read kernel function: %s", 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 >= EP)
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_rng_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_rng_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_rng_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00157         atm2->lat[atm2->np] = atm->lat[ip]
00158             + gsl_rng_gaussian_ziggurat(rng, DY2DEG(dy, atm->lon[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     /* Adjust air parcel index... */
00170     if (ctl.qnt_idx >= 0)
00171         atm2->q[ctl.qnt_idx][atm2->np] = atm2->np;
00172
00173     /* Increment particle counter... */
00174     if ((++atm2->np) > NP)
00175         ERRMSG("Too many air parcels!");
00176 }
00177
00178 /* Save data and close file... */
00179 write_atm(argv[3], &ctl, atm2, 0);
00180
00181 /* Free... */
00182 free(atm);
00183 free(atm2);
00184
00185 return EXIT_SUCCESS;
00186 }

```

## 5.11 atm\_stat.c File Reference

Calculate air parcel statistics.

```
#include "libtrac.h"
```

### 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 = GSL_NAN, 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     LOG(1, "Write air parcel statistics: %s", 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     size_t len = strlen(argv[f]);
00094     sprintf(tstr, "%.4s", &argv[f][len - 20]);
00095     year = atoi(tstr);
00096     sprintf(tstr, "%.2s", &argv[f][len - 15]);
00097     mon = atoi(tstr);
00098     sprintf(tstr, "%.2s", &argv[f][len - 12]);
00099     day = atoi(tstr);
00100     sprintf(tstr, "%.2s", &argv[f][len - 9]);
00101     hour = atoi(tstr);
00102     sprintf(tstr, "%.2s", &argv[f][len - 6]);
00103     min = atoi(tstr);
00104     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00105
00106     /* Check time... */
00107     if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00108         || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00109         ERRMSG("Cannot read time from filename!");
00110
00111     /* Save initial time... */
00112     if (!init) {
00113         init = 1;
00114         t0 = t;
00115     }
00116
00117     /* Filter data... */
00118     atm_filt->np = 0;
00119     for (ip = 0; ip < atm->np; ip++) {
00120
00121         /* Check time... */
00122         if (!gsl_finite(atm->time[ip]))
00123             continue;
00124
00125         /* Check ensemble index... */
00126         if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00127             continue;
00128
00129         /* Check spatial range... */
00130         if (atm->p[ip] > p0 || atm->p[ip] < p1
00131             || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00132             || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00133             continue;
00134
00135         /* Save data... */
00136         atm_filt->time[atm_filt->np] = atm->time[ip];
00137         atm_filt->p[atm_filt->np] = atm->p[ip];
00138         atm_filt->lon[atm_filt->np] = atm->lon[ip];
00139         atm_filt->lat[atm_filt->np] = atm->lat[ip];
00140         for (iq = 0; iq < ctl.nq; iq++)
00141             atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00142         atm_filt->np++;
00143     }
00144
00145     /* Get heights... */
00146     for (ip = 0; ip < atm_filt->np; ip++)
00147         zs[ip] = Z(atm_filt->p[ip]);
00148
00149     /* Get statistics... */
00150     if (strcasecmp(argv[3], "mean") == 0) {
00151         zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00152         lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00153         latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00154         for (iq = 0; iq < ctl.nq; iq++)
00155             qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcasecmp(argv[3], "stddev") == 0) {
00157         zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00158         lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00159         latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00160         for (iq = 0; iq < ctl.nq; iq++)
00161             qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcasecmp(argv[3], "min") == 0) {
00163         zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00164         lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00165         latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00166         for (iq = 0; iq < ctl.nq; iq++)
00167             qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcasecmp(argv[3], "max") == 0) {
00169         zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00170         lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00171         latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00172         for (iq = 0; iq < ctl.nq; iq++)
00173             qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcasecmp(argv[3], "skew") == 0) {
00175         zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);

```

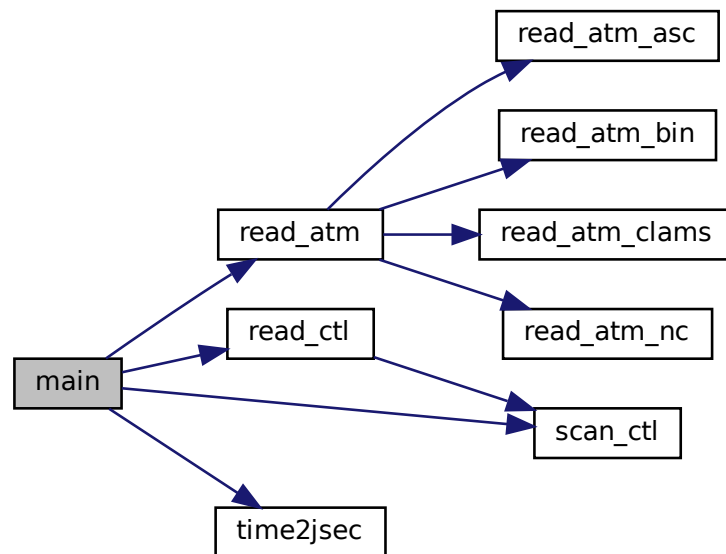
```

00176     lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00177     latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00178     for (iq = 0; iq < ctl.nq; iq++)
00179         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00180 } else if (strcasecmp(argv[3], "kurt") == 0) {
00181     zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00182     lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00183     latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00184     for (iq = 0; iq < ctl.nq; iq++)
00185         qm[iq] =
00186             gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187 } else if (strcasecmp(argv[3], "median") == 0) {
00188     zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00189     lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00190     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00191     for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193 } else if (strcasecmp(argv[3], "absdev") == 0) {
00194     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00195     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00196     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00197     for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00199 } else if (strcasecmp(argv[3], "mad") == 0) {
00200     zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00201     lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00202     latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00203     for (iq = 0; iq < ctl.nq; iq++)
00204         qm[iq] =
00205             gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00206 } else
00207     ERRMSG("Unknown parameter!");
00208
00209 /* Write data... */
00210 fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00211 for (iq = 0; iq < ctl.nq; iq++) {
00212     fprintf(out, " ");
00213     fprintf(out, ctl.qnt_format[iq], qm[iq]);
00214 }
00215 fprintf(out, " %d\n", atm_filt->np);
00216 }
00217
00218 /* Close file... */
00219 fclose(out);
00220
00221 /* Free... */
00222 free(atm);
00223 free(atm_filt);
00224 free(work);
00225 free(zs);
00226
00227 return EXIT_SUCCESS;
00228 }

```



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) 2013-2022 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 = GSL_NAN, 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> <atml> [<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     LOG(1, "Write air parcel statistics: %s", 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         size_t len = strlen(argv[f]);
00094         sprintf(tstr, "%.4s", &argv[f][len - 20]);
00095         year = atoi(tstr);
00096         sprintf(tstr, "%.2s", &argv[f][len - 15]);
00097         mon = atoi(tstr);
00098         sprintf(tstr, "%.2s", &argv[f][len - 12]);
00099         day = atoi(tstr);
00100         sprintf(tstr, "%.2s", &argv[f][len - 9]);
00101         hour = atoi(tstr);
00102         sprintf(tstr, "%.2s", &argv[f][len - 6]);
00103         min = atoi(tstr);
00104         time2jsec(year, mon, day, hour, min, 0, 0, &t);
00105
00106         /* Check time... */
00107         if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00108             || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00109             ERRMSG("Cannot read time from filename!");
00110
00111         /* Save initial time... */
00112         if (!init) {
00113             init = 1;
00114             t0 = t;
00115         }
00116
00117         /* Filter data... */
00118         atm_filt->np = 0;
00119         for (ip = 0; ip < atm->np; ip++) {
00120
00121             /* Check time... */
00122             if (!gsl_finite(atm->time[ip]))
00123                 continue;
00124
00125             /* Check ensemble index... */
00126             if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00127                 continue;
00128
00129             /* Check spatial range... */
00130             if (atm->p[ip] > p0 || atm->p[ip] < p1
00131                 || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00132                 || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00133                 continue;
00134

```

```

00135      /* Save data... */
00136      atm_filt->time[atm_filt->np] = atm->time[ip];
00137      atm_filt->p[atm_filt->np] = atm->p[ip];
00138      atm_filt->lon[atm_filt->np] = atm->lon[ip];
00139      atm_filt->lat[atm_filt->np] = atm->lat[ip];
00140      for (iq = 0; iq < ctl.nq; iq++)
00141          atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00142      atm_filt->np++;
00143  }
00144
00145      /* Get heights... */
00146      for (ip = 0; ip < atm_filt->np; ip++)
00147          zs[ip] = Z(atm_filt->p[ip]);
00148
00149      /* Get statistics... */
00150      if (strcmp(argv[3], "mean") == 0) {
00151          zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00152          lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00153          latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00154          for (iq = 0; iq < ctl.nq; iq++)
00155              qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156      } else if (strcmp(argv[3], "stddev") == 0) {
00157          zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00158          lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00159          latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00160          for (iq = 0; iq < ctl.nq; iq++)
00161              qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162      } else if (strcmp(argv[3], "min") == 0) {
00163          zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00164          lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00165          latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00166          for (iq = 0; iq < ctl.nq; iq++)
00167              qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168      } else if (strcmp(argv[3], "max") == 0) {
00169          zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00170          lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00171          latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00172          for (iq = 0; iq < ctl.nq; iq++)
00173              qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174      } else if (strcmp(argv[3], "skew") == 0) {
00175          zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00176          lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00177          latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00178          for (iq = 0; iq < ctl.nq; iq++)
00179              qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00180      } else if (strcmp(argv[3], "kurt") == 0) {
00181          zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00182          lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00183          latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00184          for (iq = 0; iq < ctl.nq; iq++)
00185              qm[iq] =
00186                  gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187      } else if (strcmp(argv[3], "median") == 0) {
00188          zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00189          lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00190          latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00191          for (iq = 0; iq < ctl.nq; iq++)
00192              qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193      } else if (strcmp(argv[3], "absdev") == 0) {
00194          zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00195          lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00196          latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00197          for (iq = 0; iq < ctl.nq; iq++)
00198              qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00199      } else if (strcmp(argv[3], "mad") == 0) {
00200          zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00201          lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00202          latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00203          for (iq = 0; iq < ctl.nq; iq++)
00204              qm[iq] =
00205                  gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00206      } else
00207          ERRMSG("Unknown parameter!");
00208
00209      /* Write data... */
00210      fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00211      for (iq = 0; iq < ctl.nq; iq++) {
00212          fprintf(out, " ");
00213          fprintf(out, ctl.qnt_format[iq], qm[iq]);
00214      }
00215      fprintf(out, " %d\n", atm_filt->np);
00216  }
00217
00218      /* Close file... */
00219      fclose(out);
00220
00221      /* Free... */

```

```
00222     free(atm);
00223     free(atm_filt);
00224     free(work);
00225     free(zs);
00226
00227     return EXIT_SUCCESS;
00228 }
```

## 5.13 day2doy.c File Reference

Convert date to day of year.

```
#include "libtrac.h"
```

### 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

Convert day of year to date.

```
#include "libtrac.h"
```

### 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     doy2day(year, doy, &mon, &day);
00043     printf("%d %d %d\n", year, mon, day);
00044     return EXIT_SUCCESS;
00045 }
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
00020 #include "libtrac.h"
00021
00022 int main(
00023     int argc,
00024     char *argv[]) {
00025
00026     int day, doy, mon, year;
00027
00028     /* Check arguments... */
00029     if (argc < 3)
00030         ERRMSG("Give parameters: <year> <doy>");
00031
00032     /* Read arguments... */
00033     year = atoi(argv[1]);
00034     doy = atoi(argv[2]);
00035
00036     /* Convert... */
00037     doy2day(year, doy, &mon, &day);
00038     printf("%d %d %d\n", year, mon, day);
00039     return EXIT_SUCCESS;
00040 }
00041 }
```

## 5.17 jsec2time.c File Reference

Convert Julian seconds to date.

```
#include "libtrac.h"
```

### 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 libtrac.c File Reference

MPTRAC library definitions.

```
#include "libtrac.h"
```

### Functions

- void [cart2geo](#) (double \*x, double \*z, double \*lon, double \*lat)  
*Convert Cartesian coordinates to geolocation.*
- double [clim\\_hno3](#) ([clim\\_t](#) \*clim, double t, double lat, double p)  
*Climatology of HNO<sub>3</sub> volume mixing ratios.*
- void [clim\\_hno3\\_init](#) ([clim\\_t](#) \*clim)  
*Initialization function for HNO<sub>3</sub> climatology.*
- double [clim\\_oh](#) ([clim\\_t](#) \*clim, double t, double lat, double p)  
*Climatology of OH number concentrations.*
- double [clim\\_oh\\_diurnal](#) ([ctl\\_t](#) \*ctl, [clim\\_t](#) \*clim, double t, double p, double lon, double lat)  
*Climatology of OH number concentrations with diurnal variation.*
- void [clim\\_oh\\_init](#) ([ctl\\_t](#) \*ctl, [clim\\_t](#) \*clim)  
*Initialization function for OH climatology.*
- double [clim\\_oh\\_init\\_help](#) (double beta, double time, double lat)  
*Apply diurnal correction to OH climatology.*



- double `clim_h2o2` (`clim_t` \*clim, double t, double lat, double p)  
*Climatology of H2O2 number concentrations.*
- void `clim_h2o2_init` (`ctl_t` \*ctl, `clim_t` \*clim)  
*Initialization function for H2O2 climatology.*
- double `clim_tropo` (`clim_t` \*clim, double t, double lat)  
*Climatology of tropopause pressure.*
- void `clim_tropo_init` (`clim_t` \*clim)  
*Initialize tropopause climatology.*
- void `compress_pack` (char \*varname, float \*array, size\_t nxy, size\_t nz, int decompress, FILE \*inout)  
*Pack or unpack array.*
- void `day2doy` (int year, int mon, int day, int \*doy)  
*Compress or decompress array with zfp.*
- 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, `clim_t` \*clim, double t, `met_t` \*\*met0, `met_t` \*\*met1)  
*Get meteo data for given time step.*
- void `get_met_help` (`ctl_t` \*ctl, double t, int direct, char \*metbase, double dt\_met, char \*filename)  
*Get meteo 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 meteo 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 meteo 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)  
*Spatial interpolation of meteo 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 meteo data.*
- void `jsec2time` (double jsec, int \*year, int \*mon, int \*day, int \*hour, int \*min, int \*sec, double \*remain)  
*Temporal interpolation of meteo data.*
- 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.*
- void `quicksort` (double arr[], int brr[], int low, int high)  
*Parallel quicksort.*
- int `quicksort_partition` (double arr[], int brr[], int low, int high)  
*Partition function for quicksort.*
- int `read_atm` (const char \*filename, `ctl_t` \*ctl, `atm_t` \*atm)  
*Read atmospheric data.*
- int `read_atm_asc` (const char \*filename, `ctl_t` \*ctl, `atm_t` \*atm)

- Read atmospheric data in ASCII format.*
- int `read_atm_bin` (const char \*filename, `ctl_t` \*ctl, `atm_t` \*atm)
- Read atmospheric data in binary format.*
- int `read_atm_clams` (const char \*filename, `ctl_t` \*ctl, `atm_t` \*atm)
- Read atmospheric data in CLaMS format.*
- int `read_atm_nc` (const char \*filename, `ctl_t` \*ctl, `atm_t` \*atm)
- Read atmospheric data in netCDF format.*
- void `read_clim` (`ctl_t` \*ctl, `clim_t` \*clim)
- Read climatological data.*
- void `read_ctl` (const char \*filename, int argc, char \*argv[], `ctl_t` \*ctl)
- Read control parameters.*
- int `read_met` (char \*filename, `ctl_t` \*ctl, `clim_t` \*clim, `met_t` \*met)
- Read meteo data file.*
- void `read_met_bin_2d` (FILE \*in, `met_t` \*met, float var[EX][EY], char \*varname)
- Read 2-D meteo variable.*
- void `read_met_bin_3d` (FILE \*in, `ctl_t` \*ctl, `met_t` \*met, float var[EX][EY][EP], char \*varname, int precision, double tolerance)
- Read 3-D meteo variable.*
- void `read_met_cape` (`clim_t` \*clim, `met_t` \*met)
- Calculate convective available potential energy.*
- void `read_met_cloud` (`ctl_t` \*ctl, `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 meteo 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 meteo data.*
- void `read_met_levels` (int ncid, `ctl_t` \*ctl, `met_t` \*met)
- Read meteo data on vertical levels.*
- void `read_met_ml2pl` (`ctl_t` \*ctl, `met_t` \*met, float var[EX][EY][EP])
- Convert meteo data from model levels to pressure levels.*
- int `read_met_nc_2d` (int ncid, char \*varname, char \*varname2, `ctl_t` \*ctl, `met_t` \*met, float dest[EX][EY], float scl, int init)
- Read and convert 2D variable from meteo data file.*
- int `read_met_nc_3d` (int ncid, char \*varname, char \*varname2, `ctl_t` \*ctl, `met_t` \*met, float dest[EX][EY][EP], float scl, int init)
- Read and convert 3D variable from meteo data file.*
- void `read_met_pbl` (`met_t` \*met)
- Calculate pressure of the boundary layer.*
- void `read_met_periodic` (`met_t` \*met)
- Create meteo 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 meteo data.*
- void `read_met_surface` (int ncid, `met_t` \*met, `ctl_t` \*ctl)
- Read surface data.*
- void `read_met_tropo` (`ctl_t` \*ctl, `clim_t` \*clim, `met_t` \*met)

- Calculate tropopause data.*
- void `read_obs` (char \*filename, double \*rt, double \*rz, double \*rlon, double \*rlat, double \*robs, int \*nobs)
- Read observation 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 rp, double rhop)
- Calculate sedimentation velocity.*
- void `spline` (double \*x, double \*y, int n, double \*x2, double \*y2, int n2, int method)
- Spline interpolation.*
- float `stddev` (float \*data, int n)
- Calculate standard deviation.*
- double `sza` (double sec, double lon, double lat)
- Calculate solar zenith angle.*
- 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, const char \*group, int output)
- Measure wall-clock time.*
- double `tropo_weight` (clim\_t \*clim, double t, double lat, double p)
- Get weighting factor based on tropopause distance.*
- void `write_atm` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm, double t)
- Write atmospheric data.*
- void `write_atm_asc` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm, double t)
- Write atmospheric data in ASCII format.*
- void `write_atm_bin` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)
- Write atmospheric data in binary format.*
- void `write_atm_clams` (ctl\_t \*ctl, atm\_t \*atm, double t)
- Write atmospheric data in CLaMS format.*
- void `write_atm_nc` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)
- Write atmospheric data in netCDF format.*
- 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_grid_asc` (const char \*filename, ctl\_t \*ctl, double \*cd, double \*vmr\_expl, double \*vmr\_impl, double t, double \*z, double \*lon, double \*lat, double \*area, double dz, int \*np)
- Write gridded data in ASCII format.*
- void `write_grid_nc` (const char \*filename, ctl\_t \*ctl, double \*cd, double \*vmr\_expl, double \*vmr\_impl, double t, double \*z, double \*lon, double \*lat, double \*area, double dz, int \*np)
- Write gridded data in netCDF format.*
- int `write_met` (char \*filename, ctl\_t \*ctl, met\_t \*met)
- Read meteo data file.*
- void `write_met_bin_2d` (FILE \*out, met\_t \*met, float var[EX][EY], char \*varname)
- Write 2-D meteo variable.*
- void `write_met_bin_3d` (FILE \*out, ctl\_t \*ctl, met\_t \*met, float var[EX][EY][EP], char \*varname, int precision, double tolerance)
- Write 3-D meteo variable.*
- 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.*

## 5.19.1 Detailed Description

MPTRAC library definitions.

Definition in file `libtrac.c`.

## 5.19.2 Function Documentation

**5.19.2.1 `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 }
```

**5.19.2.2 `clim_hno3()`** double `clim_hno3` (  
`clim_t` \* `clim`,  
double `t`,  
double `lat`,  
double `p` )

Climatology of HNO3 volume mixing ratios.

Definition at line 43 of file `libtrac.c`.

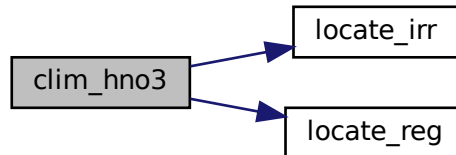
```
00047     {
00048
00049     /* Get seconds since begin of year... */
00050     double sec = FMOD(t, 365.25 * 86400.);
00051     while (sec < 0)
00052         sec += 365.25 * 86400.;
00053
00054     /* Check pressure... */
00055     if (p < clim->hno3_p[0])
00056         p = clim->hno3_p[0];
00057     else if (p > clim->hno3_p[clim->hno3_np - 1])
00058         p = clim->hno3_p[clim->hno3_np - 1];
00059
00060     /* Check latitude... */
00061     if (lat < clim->hno3_lat[0])
00062         lat = clim->hno3_lat[0];
00063     else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
```

```

00064     lat = clim->hno3_lat[clim->hno3_nlat - 1];
00065
00066     /* Get indices... */
00067     int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
00068     int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
00069     int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);
00070
00071     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00072     double aux00 = LIN(clim->hno3_p[ip],
00073                       clim->hno3[isec][ilat][ip],
00074                       clim->hno3_p[ip + 1],
00075                       clim->hno3[isec][ilat][ip + 1], p);
00076     double aux01 = LIN(clim->hno3_p[ip],
00077                       clim->hno3[isec][ilat + 1][ip],
00078                       clim->hno3_p[ip + 1],
00079                       clim->hno3[isec][ilat + 1][ip + 1], p);
00080     double aux10 = LIN(clim->hno3_p[ip],
00081                       clim->hno3[isec + 1][ilat][ip],
00082                       clim->hno3_p[ip + 1],
00083                       clim->hno3[isec + 1][ilat][ip + 1], p);
00084     double aux11 = LIN(clim->hno3_p[ip],
00085                       clim->hno3[isec + 1][ilat + 1][ip],
00086                       clim->hno3_p[ip + 1],
00087                       clim->hno3[isec + 1][ilat + 1][ip + 1], p);
00088     aux00 = LIN(clim->hno3_lat[ilat], aux00,
00089               clim->hno3_lat[ilat + 1], aux01, lat);
00090     aux11 = LIN(clim->hno3_lat[ilat], aux10,
00091               clim->hno3_lat[ilat + 1], aux11, lat);
00092     aux00 = LIN(clim->hno3_time[isec], aux00,
00093               clim->hno3_time[isec + 1], aux11, sec);
00094
00095     /* Convert from ppb to ppv... */
00096     return GSL_MAX(1e-9 * aux00, 0.0);
00097 }

```

Here is the call graph for this function:



**5.19.2.3 clim\_hno3\_init()** void clim\_hno3\_init (  
     clim\_t \* clim )

Initialization function for HNO3 climatology.

Definition at line 101 of file libtrac.c.

```

00102     {
00103
00104     /* Write info... */
00105     LOG(1, "Initialize HNO3 data...");
00106
00107     clim->hno3_ntime = 12;
00108     double hno3_time[12] = {
00109         1209600.00, 3888000.00, 6393600.00,
00110         9072000.00, 11664000.00, 14342400.00,
00111         16934400.00, 19612800.00, 22291200.00,
00112         24883200.00, 27561600.00, 30153600.00
00113     };
00114     memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));

```

```

00115
00116 clim->hno3_nlat = 18;
00117 double hno3_lat[18] = {
00118     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00119     5, 15, 25, 35, 45, 55, 65, 75, 85
00120 };
00121 memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));
00122
00123 clim->hno3_np = 10;
00124 double hno3_p[10] = {
00125     4.64159, 6.81292, 10, 14.678, 21.5443,
00126     31.6228, 46.4159, 68.1292, 100, 146.78
00127 };
00128 memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));
00129
00130 double hno3[12][18][10] = {
00131     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00132      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00133      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00134      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00135      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00136      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00137      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00138      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00139      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00140      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00141      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00142      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00143      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00144      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00145      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00146      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00147      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00148      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00149     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00150      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00151      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00152      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00153      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00154      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00155      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00156      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00157      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00158      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00159      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00160      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00161      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00162      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00163      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00164      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00165      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00166      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00167     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00168      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00169      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00170      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00171      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00172      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00173      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00174      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00175      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00176      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00177      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00178      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00179      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00180      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00181      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00182      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00183      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00184      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00185     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00186      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00187      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00188      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00189      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00190      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00191      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00192      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00193      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00194      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00195      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00196      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00197      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00198      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00199      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00200      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00201      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},

```

```

00202     {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00203     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63}},
00204     {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57}},
00205     {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63}},
00206     {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37}},
00207     {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}},
00208     {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527}},
00209     {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229}},
00210     {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972}},
00211     {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126}},
00212     {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183}},
00213     {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18}},
00214     {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343}},
00215     {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964}},
00216     {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83}},
00217     {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25}},
00218     {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39}},
00219     {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52}},
00220     {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00221     {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26}},
00222     {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05}},
00223     {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65}},
00224     {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67}},
00225     {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13}},
00226     {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639}},
00227     {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217}},
00228     {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694}},
00229     {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136}},
00230     {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}},
00231     {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229}},
00232     {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302}},
00233     {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66}},
00234     {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41}},
00235     {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8}},
00236     {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9}},
00237     {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88}},
00238     {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00239     {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}},
00240     {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78}},
00241     {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08}},
00242     {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}},
00243     {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38}},
00244     {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}},
00245     {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176}},
00246     {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705}},
00247     {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}},
00248     {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199}},
00249     {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}},
00250     {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259}},
00251     {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422}},
00252     {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913}},
00253     {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4}},
00254     {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56}},
00255     {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61}},
00256     {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00257     {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}},
00258     {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}},
00259     {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6}},
00260     {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}},
00261     {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}},
00262     {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672}},
00263     {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19}},
00264     {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181}},
00265     {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107}},
00266     {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185}},
00267     {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224}},
00268     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232}},
00269     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341}},
00270     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754}},
00271     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23}},
00272     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45}},
00273     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5}},
00274     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00275     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56}},
00276     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74}},
00277     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09}},
00278     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83}},
00279     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}},
00280     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646}},
00281     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169}},
00282     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587}},
00283     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815}},
00284     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147}},
00285     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197}},
00286     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163}},
00287     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303}},
00288     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}},

```

```

00289     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00290     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00291     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00292     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00293     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00294     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00295     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00296     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00297     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00298     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00299     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00300     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00301     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00302     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00303     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00304     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00305     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00306     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00307     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00308     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00309     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00310     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00311     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00312     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00313     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00314     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00315     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00316     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00317     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00318     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00319     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00320     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00321     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00322     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00323     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00324     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00325     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00326     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00327     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00328     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00329     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00330     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00331     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00332     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00333     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00334     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00335     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00336     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00337     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00338     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00339     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00340     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00341     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00342     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00343     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00344     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00345     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00346     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00347 };
00348 memcpy(clim->hno3, hno3, sizeof(clim->hno3));
00349
00350 /* Get range... */
00351 double hno3min = 1e99, hno3max = -1e99;
00352 for (int it = 0; it < clim->hno3_ntime; it++)
00353     for (int iz = 0; iz < clim->hno3_np; iz++)
00354         for (int iy = 0; iy < clim->hno3_nlat; iy++) {
00355             hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
00356             hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
00357         }
00358
00359 /* Write info... */
00360 LOG(2, "Number of time steps: %d", clim->hno3_ntime);
00361 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00362     clim->hno3_time[0], clim->hno3_time[1],
00363     clim->hno3_time[clim->hno3_ntime - 1]);
00364 LOG(2, "Number of pressure levels: %d", clim->hno3_np);
00365 LOG(2, "Altitude levels: %g, %g ... %g km",
00366     Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
00367     Z(clim->hno3_p[clim->hno3_np - 1]));
00368 LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
00369     clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
00370 LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
00371 LOG(2, "Latitudes: %g, %g ... %g deg",
00372     clim->hno3_lat[0], clim->hno3_lat[1],
00373     clim->hno3_lat[clim->hno3_nlat - 1]);
00374 LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
00375     1e-9 * hno3max);

```



```
00376 }
```

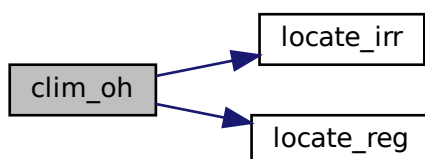
**5.19.2.4 clim\_oh()** double clim\_oh (  
     clim\_t \* clim,  
     double t,  
     double lat,  
     double p )

Climatology of OH number concentrations.

Definition at line 380 of file libtrac.c.

```
00384     {
00385
00386     /* Get seconds since begin of year... */
00387     double sec = FMOD(t, 365.25 * 86400.);
00388     while (sec < 0)
00389         sec += 365.25 * 86400.;
00390
00391     /* Check pressure... */
00392     if (p < clim->oh_p[clim->oh_np - 1])
00393         p = clim->oh_p[clim->oh_np - 1];
00394     else if (p > clim->oh_p[0])
00395         p = clim->oh_p[0];
00396
00397     /* Check latitude... */
00398     if (lat < clim->oh_lat[0])
00399         lat = clim->oh_lat[0];
00400     else if (lat > clim->oh_lat[clim->oh_nlat - 1])
00401         lat = clim->oh_lat[clim->oh_nlat - 1];
00402
00403     /* Get indices... */
00404     int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
00405     int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
00406     int ip = locate_irr(clim->oh_p, clim->oh_np, p);
00407
00408     /* Interpolate OH climatology... */
00409     double aux00 = LIN(clim->oh_p[ip],
00410                       clim->oh[isec][ip][ilat],
00411                       clim->oh_p[ip + 1],
00412                       clim->oh[isec][ip + 1][ilat], p);
00413     double aux01 = LIN(clim->oh_p[ip],
00414                       clim->oh[isec][ip][ilat + 1],
00415                       clim->oh_p[ip + 1],
00416                       clim->oh[isec][ip + 1][ilat + 1], p);
00417     double aux10 = LIN(clim->oh_p[ip],
00418                       clim->oh[isec + 1][ip][ilat],
00419                       clim->oh_p[ip + 1],
00420                       clim->oh[isec + 1][ip + 1][ilat], p);
00421     double aux11 = LIN(clim->oh_p[ip],
00422                       clim->oh[isec + 1][ip][ilat + 1],
00423                       clim->oh_p[ip + 1],
00424                       clim->oh[isec + 1][ip + 1][ilat + 1], p);
00425     aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
00426     aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
00427     aux00 =
00428         LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);
00429
00430     return GSL_MAX(aux00, 0.0);
00431 }
```

Here is the call graph for this function:



```

5.19.2.5 clim_oh_diurnal() double clim_oh_diurnal (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    double p,
    double lon,
    double lat )

```

Climatology of OH number concentrations with diurnal variation.

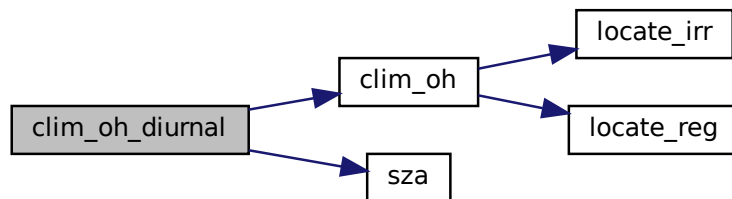
Definition at line 435 of file `libtrac.c`.

```

00441     {
00442
00443     double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);
00444
00445     if (sza2 <= M_PI / 2. * 89. / 90.)
00446         return oh * exp(-ctl->oh_chem_beta / cos(sza2));
00447     else
00448         return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
00449 }

```

Here is the call graph for this function:



```

5.19.2.6 clim_oh_init() void clim_oh_init (
    ctl_t * ctl,
    clim_t * clim )

```

Initialization function for OH climatology.

Definition at line 453 of file `libtrac.c`.

```

00455     {
00456
00457     int nt, ncid, varid;
00458
00459     double *help, ohmin = 1e99, ohmax = -1e99;
00460
00461     /* Write info... */
00462     LOG(1, "Read OH data: %s", ctl->clim_oh_filename);
00463
00464     /* Open netCDF file... */
00465     if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00466         WARN("OH climatology data are missing!");
00467         return;
00468     }
00469 }

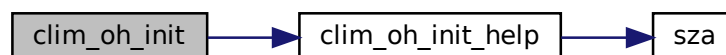
```

```

00470  /* Read pressure data... */
00471  NC_INQ_DIM("press", &clim->oh_np, 2, CP);
00472  NC_GET_DOUBLE("press", clim->oh_p, 1);
00473
00474  /* Check ordering of pressure data... */
00475  if (clim->oh_p[0] < clim->oh_p[1])
00476      ERRMSG("Pressure data are not descending!");
00477
00478  /* Read latitudes... */
00479  NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
00480  NC_GET_DOUBLE("lat", clim->oh_lat, 1);
00481
00482  /* Check ordering of latitudes... */
00483  if (clim->oh_lat[0] > clim->oh_lat[1])
00484      ERRMSG("Latitude data are not ascending!");
00485
00486  /* Set time data for monthly means... */
00487  clim->oh_ntime = 12;
00488  clim->oh_time[0] = 1209600.00;
00489  clim->oh_time[1] = 3888000.00;
00490  clim->oh_time[2] = 6393600.00;
00491  clim->oh_time[3] = 9072000.00;
00492  clim->oh_time[4] = 11664000.00;
00493  clim->oh_time[5] = 14342400.00;
00494  clim->oh_time[6] = 16934400.00;
00495  clim->oh_time[7] = 19612800.00;
00496  clim->oh_time[8] = 22291200.00;
00497  clim->oh_time[9] = 24883200.00;
00498  clim->oh_time[10] = 27561600.00;
00499  clim->oh_time[11] = 30153600.00;
00500
00501  /* Check number of timesteps... */
00502  NC_INQ_DIM("time", &nt, 12, 12);
00503
00504  /* Read OH data... */
00505  ALLOC(help, double,
00506        clim->oh_nlat * clim->oh_np * clim->oh_ntime);
00507  NC_GET_DOUBLE("OH", help, 1);
00508  for (int it = 0; it < clim->oh_ntime; it++)
00509      for (int iz = 0; iz < clim->oh_np; iz++)
00510          for (int iy = 0; iy < clim->oh_nlat; iy++) {
00511              clim->oh[it][iz][iy] =
00512                  help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
00513                  / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
00514                                     clim->oh_lat[iy]);
00515              ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
00516              ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
00517          }
00518  free(help);
00519
00520  /* Close netCDF file... */
00521  NC(nc_close(ncid));
00522
00523  /* Write info... */
00524  LOG(2, "Number of time steps: %d", clim->oh_ntime);
00525  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00526        clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
00527  LOG(2, "Number of pressure levels: %d", clim->oh_np);
00528  LOG(2, "Altitude levels: %g, %g ... %g km",
00529        Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
00530  LOG(2, "Pressure levels: %g, %g ... %g hPa",
00531        clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
00532  LOG(2, "Number of latitudes: %d", clim->oh_nlat);
00533  LOG(2, "Latitudes: %g, %g ... %g deg",
00534        clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
00535  LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
00536 }

```

Here is the call graph for this function:



```

5.19.2.7 clim_oh_init_help() double clim_oh_init_help (
    double beta,
    double time,
    double lat )

```

Apply diurnal correction to OH climatology.

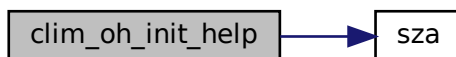
Definition at line 540 of file libtrac.c.

```

00543     {
00544
00545     double aux, lon, sum = 0;
00546
00547     int n = 0;
00548
00549     /* Integrate day/night correction factor over longitude... */
00550     for (lon = -180; lon < 180; lon += 1) {
00551         aux = sza(time, lon, lat);
00552         if (aux <= M_PI / 2. * 85. / 90.)
00553             sum += exp(-beta / cos(aux));
00554         else
00555             sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
00556         n++;
00557     }
00558     return sum / (double) n;
00559 }

```

Here is the call graph for this function:



```

5.19.2.8 clim_h2o2() double clim_h2o2 (
    clim_t * clim,
    double t,
    double lat,
    double p )

```

Climatology of H2O2 number concentrations.

Definition at line 563 of file libtrac.c.

```

00567     {
00568
00569     /* Get seconds since begin of year... */
00570     double sec = FMOD(t, 365.25 * 86400.);
00571     while (sec < 0)
00572         sec += 365.25 * 86400.;
00573
00574     /* Check pressure... */
00575     if (p < clim->h2o2_p[clim->h2o2_np - 1])
00576         p = clim->h2o2_p[clim->h2o2_np - 1];
00577     else if (p > clim->h2o2_p[0])
00578         p = clim->h2o2_p[0];
00579
00580     /* Check latitude... */
00581     if (lat < clim->h2o2_lat[0])
00582         lat = clim->h2o2_lat[0];
00583     else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
00584         lat = clim->h2o2_lat[clim->h2o2_nlat - 1];
00585 }

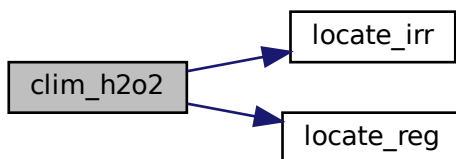
```

```

00586  /* Get indices... */
00587  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
00588  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
00589  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);
00590
00591  /* Interpolate H2O2 climatology... */
00592  double aux00 = LIN(clim->h2o2_p[ip],
00593                    clim->h2o2[isec][ip][ilat],
00594                    clim->h2o2_p[ip + 1],
00595                    clim->h2o2[isec][ip + 1][ilat], p);
00596  double aux01 = LIN(clim->h2o2_p[ip],
00597                    clim->h2o2[isec][ip][ilat + 1],
00598                    clim->h2o2_p[ip + 1],
00599                    clim->h2o2[isec][ip + 1][ilat + 1], p);
00600  double aux10 = LIN(clim->h2o2_p[ip],
00601                    clim->h2o2[isec + 1][ip][ilat],
00602                    clim->h2o2_p[ip + 1],
00603                    clim->h2o2[isec + 1][ip + 1][ilat], p);
00604  double aux11 = LIN(clim->h2o2_p[ip],
00605                    clim->h2o2[isec + 1][ip][ilat + 1],
00606                    clim->h2o2_p[ip + 1],
00607                    clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
00608  aux00 =
00609      LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
00610  aux11 =
00611      LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
00612  aux00 =
00613      LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);
00614
00615  return GSL_MAX(aux00, 0.0);
00616 }

```

Here is the call graph for this function:



**5.19.2.9 clim\_h2o2\_init()** void clim\_h2o2\_init (

```

    ctl_t * ctl,
    clim_t * clim )

```

Initialization function for H2O2 climatology.

Definition at line 620 of file libtrac.c.

```

00622  {
00623
00624  int ncid, varid, it, iy, iz, nt;
00625
00626  double *help, h2o2min = 1e99, h2o2max = -1e99;
00627
00628  /* Write info... */
00629  LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);
00630
00631  /* Open netCDF file... */
00632  if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00633      WARN("H2O2 climatology data are missing!");
00634      return;
00635  }
00636

```

```

00637  /* Read pressure data... */
00638  NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
00639  NC_GET_DOUBLE("press", clim->h2o2_p, 1);
00640
00641  /* Check ordering of pressure data... */
00642  if (clim->h2o2_p[0] < clim->h2o2_p[1])
00643      ERRMSG("Pressure data are not descending!");
00644
00645  /* Read latitudes... */
00646  NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
00647  NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);
00648
00649  /* Check ordering of latitude data... */
00650  if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
00651      ERRMSG("Latitude data are not ascending!");
00652
00653  /* Set time data (for monthly means)... */
00654  clim->h2o2_ntime = 12;
00655  clim->h2o2_time[0] = 1209600.00;
00656  clim->h2o2_time[1] = 3888000.00;
00657  clim->h2o2_time[2] = 6393600.00;
00658  clim->h2o2_time[3] = 9072000.00;
00659  clim->h2o2_time[4] = 11664000.00;
00660  clim->h2o2_time[5] = 14342400.00;
00661  clim->h2o2_time[6] = 16934400.00;
00662  clim->h2o2_time[7] = 19612800.00;
00663  clim->h2o2_time[8] = 22291200.00;
00664  clim->h2o2_time[9] = 24883200.00;
00665  clim->h2o2_time[10] = 27561600.00;
00666  clim->h2o2_time[11] = 30153600.00;
00667
00668  /* Check number of timesteps... */
00669  NC_INQ_DIM("time", &nt, 12, 12);
00670
00671  /* Read data... */
00672  ALLOC(help, double,
00673        clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
00674  NC_GET_DOUBLE("h2o2", help, 1);
00675  for (it = 0; it < clim->h2o2_ntime; it++)
00676      for (iz = 0; iz < clim->h2o2_np; iz++)
00677          for (iy = 0; iy < clim->h2o2_nlat; iy++) {
00678              clim->h2o2[it][iz][iy] =
00679                  help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
00680              h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
00681              h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
00682          }
00683  free(help);
00684
00685  /* Close netCDF file... */
00686  NC(nc_close(ncid));
00687
00688  /* Write info... */
00689  LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
00690  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00691        clim->h2o2_time[0], clim->h2o2_time[1],
00692        clim->h2o2_time[clim->h2o2_ntime - 1]);
00693  LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
00694  LOG(2, "Altitude levels: %g, %g ... %g km",
00695        Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
00696        Z(clim->h2o2_p[clim->h2o2_np - 1]));
00697  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
00698        clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
00699  LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
00700  LOG(2, "Latitudes: %g, %g ... %g deg",
00701        clim->h2o2_lat[0], clim->h2o2_lat[1],
00702        clim->h2o2_lat[clim->h2o2_nlat - 1]);
00703  LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
00704 }

```

```

5.19.2.10 clim_tropo() double clim_tropo (
    clim_t * clim,
    double t,
    double lat )

```

Climatology of tropopause pressure.

Definition at line 708 of file libtrac.c.

```

00711 {

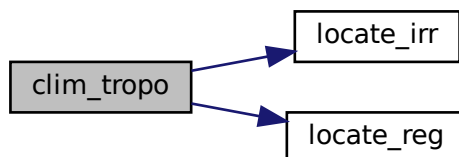
```

```

00712
00713  /* Get seconds since begin of year... */
00714  double sec = FMOD(t, 365.25 * 86400.);
00715  while (sec < 0)
00716      sec += 365.25 * 86400.;
00717
00718  /* Get indices... */
00719  int isec = locate_irr(clim->tropo_time, clim->tropo_ntime, sec);
00720  int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);
00721
00722  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
00723  double p0 = LIN(clim->tropo_lat[ilat],
00724                 clim->tropo[isec][ilat],
00725                 clim->tropo_lat[ilat + 1],
00726                 clim->tropo[isec][ilat + 1], lat);
00727  double p1 = LIN(clim->tropo_lat[ilat],
00728                 clim->tropo[isec + 1][ilat],
00729                 clim->tropo_lat[ilat + 1],
00730                 clim->tropo[isec + 1][ilat + 1], lat);
00731  return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
00732 }

```

Here is the call graph for this function:



**5.19.2.11 clim\_tropo\_init()** void clim\_tropo\_init (  
     clim\_t \* clim )

Initialize tropopause climatology.

Definition at line 736 of file libtrac.c.

```

00737  {
00738
00739  /* Write info... */
00740  LOG(1, "Initialize tropopause data...");
00741
00742  clim->tropo_ntime = 12;
00743  double tropo_time[12] = {
00744      1209600.00, 3888000.00, 6393600.00,
00745      9072000.00, 11664000.00, 14342400.00,
00746      16934400.00, 19612800.00, 22291200.00,
00747      24883200.00, 27561600.00, 30153600.00
00748  };
00749  memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));
00750
00751  clim->tropo_nlat = 73;
00752  double tropo_lat[73] = {
00753      -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00754      -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00755      -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00756      -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00757      15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00758      45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00759      75, 77.5, 80, 82.5, 85, 87.5, 90
00760  };
00761  memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));
00762

```

```
00763 double tropo[12][73] = {
00764     {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00765      297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00766      175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00767      99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00768      98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00769      152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00770      277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00771      275.3, 275.6, 275.4, 274.1, 273.5},
00772     {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00773      300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00774      150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00775      98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00776      98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00777      220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00778      284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00779      287.5, 286.2, 285.8},
00780     {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00781      297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00782      161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00783      100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00784      99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00785      186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00786      279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00787      304.3, 304.9, 306, 306.6, 306.2, 306},
00788     {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00789      290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00790      195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00791      102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00792      99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00793      148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00794      263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00795      315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00796     {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00797      260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00798      205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00799      101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00800      102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00801      165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00802      273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00803      325.3, 325.8, 325.8},
00804     {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00805      222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00806      228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00807      105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00808      106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00809      127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00810      251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00811      308.5, 312.2, 313.1, 313.3},
00812     {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00813      187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00814      235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00815      110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00816      111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00817      117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00818      224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00819      275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00820     {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00821      185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00822      233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00823      110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00824      112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00825      120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00826      230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00827      278.2, 282.6, 287.4, 290.9, 292.5, 293},
00828     {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00829      183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00830      243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00831      114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00832      110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00833      114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00834      203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00835      276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00836     {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00837      215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00838      237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00839      111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00840      106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00841      112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00842      206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00843      279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00844      305.1},
00845     {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00846      253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00847      223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00848      108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00849      102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
```



```

00850     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00851     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00852     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00853     {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00854     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00855     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00856     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00857     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00858     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00859     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00860     281.7, 281.1, 281.2}
00861 };
00862 memcpy(clim->tropo, tropo, sizeof(clim->tropo));
00863
00864 /* Get range... */
00865 double tropomin = 1e99, tropomax = -1e99;
00866 for (int it = 0; it < clim->tropo_ntime; it++)
00867     for (int iy = 0; iy < clim->tropo_nlat; iy++) {
00868         tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
00869         tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
00870     }
00871
00872 /* Write info... */
00873 LOG(2, "Number of time steps: %d", clim->tropo_ntime);
00874 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00875     clim->tropo_time[0], clim->tropo_time[1],
00876     clim->tropo_time[clim->tropo_ntime - 1]);
00877 LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
00878 LOG(2, "Latitudes: %g, %g ... %g deg",
00879     clim->tropo_lat[0], clim->tropo_lat[1],
00880     clim->tropo_lat[clim->tropo_nlat - 1]);
00881 LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
00882     Z(tropomin));
00883 LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
00884 }

```

**5.19.2.12 compress\_pack()** void compress\_pack (

```

    char * varname,
    float * array,
    size_t nxy,
    size_t nz,
    int decompress,
    FILE * inout )

```

Pack or unpack array.

Definition at line 888 of file libtrac.c.

```

00894     {
00895
00896     double min[EP], max[EP], off[EP], scl[EP];
00897
00898     unsigned short *sarray;
00899
00900     /* Allocate... */
00901     ALLOC(sarray, unsigned short,
00902         nxy * nz);
00903
00904     /* Read compressed stream and decompress array... */
00905     if (decompress) {
00906
00907         /* Write info... */
00908         LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
00909             varname, 100. * sizeof(unsigned short) / sizeof(float));
00910
00911         /* Read data... */
00912         FREAD(&scl, double,
00913             nz,
00914             inout);
00915         FREAD(&off, double,
00916             nz,
00917             inout);
00918         FREAD(sarray, unsigned short,
00919             nxy * nz,
00920             inout);
00921
00922         /* Convert to float... */

```

```

00923 #pragma omp parallel for default(shared)
00924     for (size_t ixy = 0; ixy < nxy; ixy++)
00925         for (size_t iz = 0; iz < nz; iz++)
00926             array[ixy * nz + iz]
00927                 = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
00928     }
00929
00930     /* Compress array and output compressed stream... */
00931     else {
00932
00933         /* Write info... */
00934         LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
00935             varname, 100. * sizeof(unsigned short) / sizeof(float));
00936
00937         /* Get range... */
00938         for (size_t iz = 0; iz < nz; iz++) {
00939             min[iz] = array[iz];
00940             max[iz] = array[iz];
00941         }
00942         for (size_t ixy = 1; ixy < nxy; ixy++)
00943             for (size_t iz = 0; iz < nz; iz++) {
00944                 if (array[ixy * nz + iz] < min[iz])
00945                     min[iz] = array[ixy * nz + iz];
00946                 if (array[ixy * nz + iz] > max[iz])
00947                     max[iz] = array[ixy * nz + iz];
00948             }
00949
00950         /* Get offset and scaling factor... */
00951         for (size_t iz = 0; iz < nz; iz++) {
00952             scl[iz] = (max[iz] - min[iz]) / 65533.;
00953             off[iz] = min[iz];
00954         }
00955
00956         /* Convert to short... */
00957         #pragma omp parallel for default(shared)
00958         for (size_t ixy = 0; ixy < nxy; ixy++)
00959             for (size_t iz = 0; iz < nz; iz++)
00960                 if (scl[iz] != 0)
00961                     sarray[ixy * nz + iz] = (unsigned short)
00962                         ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
00963                 else
00964                     sarray[ixy * nz + iz] = 0;
00965
00966         /* Write data... */
00967         FWRITE(&scl, double,
00968             nz,
00969             inout);
00970         FWRITE(&off, double,
00971             nz,
00972             inout);
00973         FWRITE(sarray, unsigned short,
00974             nxy * nz,
00975             inout);
00976     }
00977
00978     /* Free... */
00979     free(sarray);
00980 }

```

**5.19.2.13 day2doy()** void day2doy (

```

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

```

Compress or decompress array with zfp.

Compress or decompress array with zstd.

Get day of year from date.

Definition at line 1129 of file libtrac.c.

```

01133     {
01134
01135     const int
01136         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },

```

```

01137     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01138
01139     /* Get day of year... */
01140     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01141         *doy = d0l[mon - 1] + day - 1;
01142     else
01143         *doy = d0[mon - 1] + day - 1;
01144 }

```

#### 5.19.2.14 doym2day() void doym2day (

```

    int year,
    int doym,
    int * mon,
    int * day )

```

Get date from day of year.

Definition at line 1148 of file libtrac.c.

```

01152     {
01153
01154     const int
01155     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01156     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01157
01158     int i;
01159
01160     /* Get month and day... */
01161     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01162         for (i = 11; i > 0; i--)
01163             if (d0l[i] <= doym)
01164                 break;
01165         *mon = i + 1;
01166         *day = doym - d0l[i] + 1;
01167     } else {
01168         for (i = 11; i > 0; i--)
01169             if (d0[i] <= doym)
01170                 break;
01171         *mon = i + 1;
01172         *day = doym - d0[i] + 1;
01173     }
01174 }

```

#### 5.19.2.15 geo2cart() void geo2cart (

```

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

```

Convert geolocation to Cartesian coordinates.

Definition at line 1178 of file libtrac.c.

```

01182     {
01183
01184     double radius = z + RE;
01185     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01186     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01187     x[2] = radius * sin(lat / 180. * M_PI);
01188 }

```

```

5.19.2.16 get_met() void get_met (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    met_t ** met0,
    met_t ** met1 )

```

Get meteo data for given time step.

Definition at line 1192 of file libtrac.c.

```

01197     {
01198
01199     static int init;
01200
01201     met_t *mets;
01202
01203     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01204
01205     /* Set timer... */
01206     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01207
01208     /* Init... */
01209     if (t == ctl->t_start || !init) {
01210         init = 1;
01211
01212         /* Read meteo data... */
01213         get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
01214                     ctl->metbase, ctl->dt_met, filename);
01215         if (!read_met(filename, ctl, clim, *met0))
01216             ERRMSG("Cannot open file!");
01217
01218         get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
01219                     ctl->metbase, ctl->dt_met, filename);
01220         if (!read_met(filename, ctl, clim, *met1))
01221             ERRMSG("Cannot open file!");
01222
01223         /* Update GPU... */
01224 #ifdef _OPENACC
01225         met_t *met0up = *met0;
01226         met_t *met1up = *met1;
01227 #ifdef ASYNCIO
01228 #pragma acc update device(met0up[:1],met1up[:1]) async(5)
01229 #else
01230 #pragma acc update device(met0up[:1],met1up[:1])
01231 #endif
01232 #endif
01233
01234         /* Caching... */
01235         if (ctl->met_cache && t != ctl->t_stop) {
01236             get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
01237                         ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
01238             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01239             LOG(1, "Caching: %s", cachefile);
01240             if (system(cmd) != 0)
01241                 WARN("Caching command failed!");
01242         }
01243     }
01244
01245     /* Read new data for forward trajectories... */
01246     if (t > (*met1)->time) {
01247
01248         /* Pointer swap... */
01249         mets = *met1;
01250         *met1 = *met0;
01251         *met0 = mets;
01252
01253         /* Read new meteo data... */
01254         get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
01255         if (!read_met(filename, ctl, clim, *met1))
01256             ERRMSG("Cannot open file!");
01257         /* Update GPU... */
01258 #ifdef _OPENACC
01259         met_t *met1up = *met1;
01260 #ifdef ASYNCIO
01261 #pragma acc update device(met1up[:1]) async(5)
01262 #else
01263 #pragma acc update device(met1up[:1])
01264 #endif
01265 #endif
01266         /* Caching... */
01267         if (ctl->met_cache && t != ctl->t_stop) {
01268             get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
01269                         cachefile);

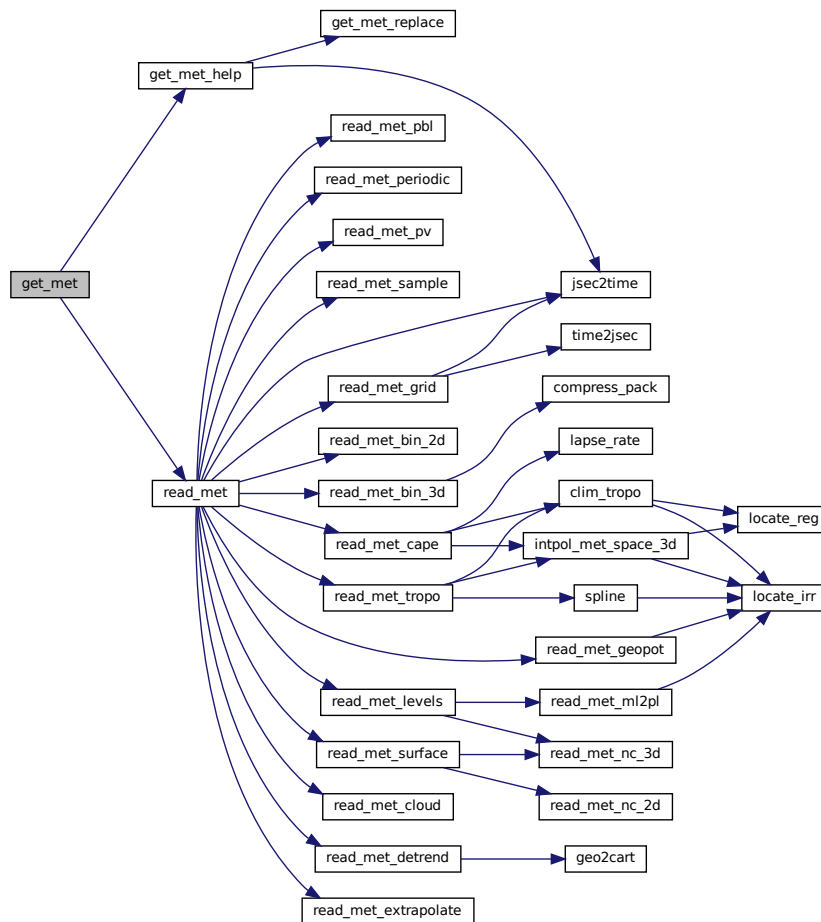
```

```

01270     sprintf(cmd, "cat %s > /dev/null &", cachefile);
01271     LOG(1, "Caching: %s", cachefile);
01272     if (system(cmd) != 0)
01273         WARN("Caching command failed!");
01274 }
01275 }
01276 /* Read new data for backward trajectories... */
01277 if (t < (*met0)->time) {
01278
01279     /* Pointer swap... */
01280     mets = *met1;
01281     *met1 = *met0;
01282     *met0 = mets;
01283
01284     /* Read new meteo data... */
01285     get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
01286     if (!read_met(filename, ctl, clim, *met0))
01287         ERRMSG("Cannot open file!");
01288
01289     /* Update GPU... */
01290 #ifdef _OPENACC
01291     met_t *met0up = *met0;
01292 #ifdef ASYNCIO
01293     #pragma acc update device(met0up[:1]) async(5)
01294 #else
01295     #pragma acc update device(met0up[:1])
01296 #endif
01297 #endif
01298
01299     /* Caching... */
01300     if (ctl->met_cache && t != ctl->t_stop) {
01301         get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
01302             cachefile);
01303         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01304         LOG(1, "Caching: %s", cachefile);
01305         if (system(cmd) != 0)
01306             WARN("Caching command failed!");
01307     }
01308 }
01309 /* Check that grids are consistent... */
01310 if ((*met0)->nx != 0 && (*met1)->nx != 0) {
01311     if ((*met0)->nx != (*met1)->nx
01312         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01313         ERRMSG("Meteo grid dimensions do not match!");
01314     for (int ix = 0; ix < (*met0)->nx; ix++)
01315         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01316             ERRMSG("Meteo grid longitudes do not match!");
01317     for (int iy = 0; iy < (*met0)->ny; iy++)
01318         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01319             ERRMSG("Meteo grid latitudes do not match!");
01320     for (int ip = 0; ip < (*met0)->np; ip++)
01321         if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01322             ERRMSG("Meteo grid pressure levels do not match!");
01323 }
01324 }

```

Here is the call graph for this function:



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

```

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

```

Get meteo data for time step.

Definition at line 1328 of file libtrac.c.

```

01334     {
01335
01336     char repl[LEN];
01337
01338     double t6, r;
01339
01340     int year, mon, day, hour, min, sec;
01341
01342     /* Round time to fixed intervals... */
01343     if (direct == -1)
01344         t6 = floor(t / dt_met) * dt_met;

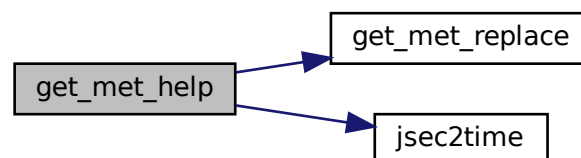
```

```

01345     else
01346         t6 = ceil(t / dt_met) * dt_met;
01347
01348     /* Decode time... */
01349     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01350
01351     /* Set filename of MPTRAC meteo files... */
01352     if (ctl->clams_met_data == 0) {
01353         if (ctl->met_type == 0)
01354             sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01355         else if (ctl->met_type == 1)
01356             sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
01357         else if (ctl->met_type == 2)
01358             sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
01359         else if (ctl->met_type == 3)
01360             sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
01361         else if (ctl->met_type == 4)
01362             sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", metbase);
01363         sprintf(repl, "%d", year);
01364         get_met_replace(filename, "YYYY", repl);
01365         sprintf(repl, "%02d", mon);
01366         get_met_replace(filename, "MM", repl);
01367         sprintf(repl, "%02d", day);
01368         get_met_replace(filename, "DD", repl);
01369         sprintf(repl, "%02d", hour);
01370         get_met_replace(filename, "HH", repl);
01371     }
01372
01373     /* Set filename of CLaMS meteo files... */
01374     else {
01375         sprintf(filename, "%s_YYMMDDHH.nc", metbase);
01376         sprintf(repl, "%d", year);
01377         get_met_replace(filename, "YYYY", repl);
01378         sprintf(repl, "%d", year % 100);
01379         get_met_replace(filename, "YY", repl);
01380         sprintf(repl, "%02d", mon);
01381         get_met_replace(filename, "MM", repl);
01382         sprintf(repl, "%02d", day);
01383         get_met_replace(filename, "DD", repl);
01384         sprintf(repl, "%02d", hour);
01385         get_met_replace(filename, "HH", repl);
01386     }
01387 }

```

Here is the call graph for this function:



**5.19.2.18 get\_met\_replace()** void get\_met\_replace (  
char \* orig,  
char \* search,  
char \* repl )

Replace template strings in filename.

Definition at line 1391 of file libtrac.c.

```

01394     {

```

```

01395
01396 char buffer[LEN];
01397
01398 /* Iterate... */
01399 for (int i = 0; i < 3; i++) {
01400
01401     /* Replace sub-string... */
01402     char *ch;
01403     if (!(ch = strstr(orig, search)))
01404         return;
01405     strncpy(buffer, orig, (size_t) (ch - orig));
01406     buffer[ch - orig] = 0;
01407     sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01408     orig[0] = 0;
01409     strcpy(orig, buffer);
01410 }
01411 }

```

### 5.19.2.19 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 meteo data.

Definition at line 1415 of file libtrac.c.

```

01424     {
01425
01426     /* Initialize interpolation... */
01427     if (init) {
01428
01429         /* Check longitude... */
01430         if (met->lon[met->nx - 1] > 180 && lon < 0)
01431             lon += 360;
01432
01433         /* Get interpolation indices... */
01434         ci[0] = locate_irr(met->p, met->np, p);
01435         ci[1] = locate_reg(met->lon, met->nx, lon);
01436         ci[2] = locate_reg(met->lat, met->ny, lat);
01437
01438         /* Get interpolation weights... */
01439         cw[0] = (met->p[ci[0] + 1] - p)
01440             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01441         cw[1] = (met->lon[ci[1] + 1] - lon)
01442             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01443         cw[2] = (met->lat[ci[2] + 1] - lat)
01444             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01445     }
01446
01447     /* Interpolate vertically... */
01448     double aux00 =
01449         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01450         + array[ci[1]][ci[2]][ci[0] + 1];
01451     double aux01 =
01452         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01453             array[ci[1]][ci[2] + 1][ci[0] + 1])
01454         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01455     double aux10 =
01456         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01457             array[ci[1] + 1][ci[2]][ci[0] + 1])
01458         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01459     double aux11 =
01460         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01461             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01462         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01463
01464     /* Interpolate horizontally... */
01465     aux00 = cw[2] * (aux00 - aux01) + aux01;
01466     aux11 = cw[2] * (aux10 - aux11) + aux11;

```

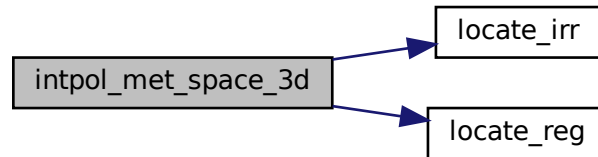


```

01467     *var = cw[1] * (aux00 - aux11) + aux11;
01468 }

```

Here is the call graph for this function:



**5.19.2.20 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 meteo data.

Definition at line 1472 of file libtrac.c.

```

01480     {
01481
01482     /* Initialize interpolation... */
01483     if (init) {
01484
01485     /* Check longitude... */
01486     if (met->lon[met->nx - 1] > 180 && lon < 0)
01487         lon += 360;
01488
01489     /* Get interpolation indices... */
01490     ci[1] = locate_reg(met->lon, met->nx, lon);
01491     ci[2] = locate_reg(met->lat, met->ny, lat);
01492
01493     /* Get interpolation weights... */
01494     cw[1] = (met->lon[ci[1] + 1] - lon)
01495         / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01496     cw[2] = (met->lat[ci[2] + 1] - lat)
01497         / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01498     }
01499
01500     /* Set variables... */
01501     double aux00 = array[ci[1]][ci[2]];
01502     double aux01 = array[ci[1]][ci[2] + 1];
01503     double aux10 = array[ci[1] + 1][ci[2]];
01504     double aux11 = array[ci[1] + 1][ci[2] + 1];
01505
01506     /* Interpolate horizontally... */
01507     if (isfinite(aux00) && isfinite(aux01)
01508         && isfinite(aux10) && isfinite(aux11)) {
01509         aux00 = cw[2] * (aux00 - aux01) + aux01;
01510         aux11 = cw[2] * (aux10 - aux11) + aux11;
01511         *var = cw[1] * (aux00 - aux11) + aux11;
01512     } else {
01513         if (cw[2] < 0.5) {
01514             if (cw[1] < 0.5)

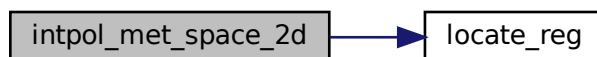
```

```

01515     *var = aux11;
01516     else
01517     *var = aux01;
01518 } else {
01519     if (cw[1] < 0.5)
01520     *var = aux10;
01521     else
01522     *var = aux00;
01523 }
01524 }
01525 }

```

Here is the call graph for this function:



**5.19.2.21 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 )

```

Spatial interpolation of meteo data.

Temporal interpolation of meteo data.

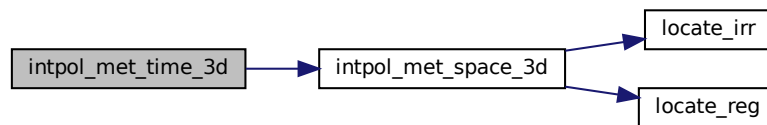
Definition at line 1632 of file libtrac.c.

```

01644     {
01645
01646     double var0, var1, wt;
01647
01648     /* Spatial interpolation... */
01649     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01650     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01651
01652     /* Get weighting factor... */
01653     wt = (met1->time - ts) / (met1->time - met0->time);
01654
01655     /* Interpolate... */
01656     *var = wt * (var0 - var1) + var1;
01657 }

```

Here is the call graph for this function:



#### 5.19.2.22 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 meteo data.

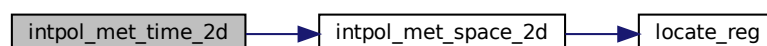
Definition at line 1661 of file libtrac.c.

```

01672     {
01673
01674     double var0, var1, wt;
01675
01676     /* Spatial interpolation... */
01677     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01678     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01679
01680     /* Get weighting factor... */
01681     wt = (met1->time - ts) / (met1->time - met0->time);
01682
01683     /* Interpolate... */
01684     if (isfinite(var0) && isfinite(var1))
01685         *var = wt * (var0 - var1) + var1;
01686     else if (wt < 0.5)
01687         *var = var1;
01688     else
01689         *var = var0;
01690 }

```

Here is the call graph for this function:



```

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

```

Temporal interpolation of meteo data.

Convert seconds to date.

Definition at line 1725 of file libtrac.c.

```

01733     {
01734
01735     struct tm t0, *t1;
01736
01737     t0.tm_year = 100;
01738     t0.tm_mon = 0;
01739     t0.tm_mday = 1;
01740     t0.tm_hour = 0;
01741     t0.tm_min = 0;
01742     t0.tm_sec = 0;
01743
01744     time_t jsec0 = (time_t) jsec + timegm(&t0);
01745     t1 = gmtime(&jsec0);
01746
01747     *year = t1->tm_year + 1900;
01748     *mon = t1->tm_mon + 1;
01749     *day = t1->tm_mday;
01750     *hour = t1->tm_hour;
01751     *min = t1->tm_min;
01752     *sec = t1->tm_sec;
01753     *remain = jsec - floor(jsec);
01754 }

```

```

5.19.2.24 lapse_rate() double lapse_rate (
    double t,
    double h2o )

```

Calculate moist adiabatic lapse rate.

Definition at line 1758 of file libtrac.c.

```

01760     {
01761
01762     /*
01763     Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01764     and water vapor volume mixing ratio [1].
01765
01766     Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01767     */
01768
01769     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01770
01771     return 1e3 * GO * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01772 }

```

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

Find array index for irregular grid.

Definition at line 1776 of file [libtrac.c](#).

```
01779     {
01780
01781     int ilo = 0;
01782     int ihi = n - 1;
01783     int i = (ihi + ilo) » 1;
01784
01785     if (xx[i] < xx[i + 1])
01786         while (ihi > ilo + 1) {
01787             i = (ihi + ilo) » 1;
01788             if (xx[i] > x)
01789                 ihi = i;
01790             else
01791                 ilo = i;
01792         } else
01793             while (ihi > ilo + 1) {
01794                 i = (ihi + ilo) » 1;
01795                 if (xx[i] <= x)
01796                     ihi = i;
01797                 else
01798                     ilo = i;
01799             }
01800
01801     return ilo;
01802 }
```

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

Find array index for regular grid.

Definition at line 1806 of file [libtrac.c](#).

```
01809     {
01810
01811     /* Calculate index... */
01812     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01813
01814     /* Check range... */
01815     if (i < 0)
01816         return 0;
01817     else if (i > n - 2)
01818         return n - 2;
01819     else
01820         return i;
01821 }
```

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

Calculate NAT existence temperature.

Definition at line 1825 of file [libtrac.c](#).

```
01828     {
01829
01830     /* Check water vapor vmr... */
```

```

01831  h2o = GSL_MAX(h2o, 0.1e-6);
01832
01833  /* Calculate T_NAT... */
01834  double p_hno3 = hno3 * p / 1.333224;
01835  double p_h2o = h2o * p / 1.333224;
01836  double a = 0.009179 - 0.00088 * log10(p_h2o);
01837  double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01838  double c = -11397.0 / a;
01839  double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
01840  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01841  if (x2 > 0)
01842      tnat = x2;
01843
01844  return tnat;
01845 }

```

**5.19.2.28 quicksort()** void quicksort (  
     double arr[],  
     int brr[],  
     int low,  
     int high )

Parallel quicksort.

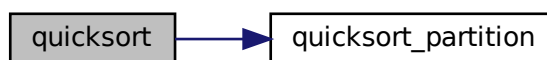
Definition at line 1849 of file libtrac.c.

```

01853  {
01854
01855  if (low < high) {
01856      int pi = quicksort_partition(arr, brr, low, high);
01857
01858  #pragma omp task firstprivate(arr,brr,low,pi)
01859      {
01860          quicksort(arr, brr, low, pi - 1);
01861      }
01862
01863      // #pragma omp task firstprivate(arr,brr,high,pi)
01864      {
01865          quicksort(arr, brr, pi + 1, high);
01866      }
01867  }
01868 }

```

Here is the call graph for this function:



**5.19.2.29 quicksort\_partition()** int quicksort\_partition (  
     double arr[],  
     int brr[],  
     int low,  
     int high )

Partition function for quicksort.

Definition at line 1872 of file libtrac.c.

```

01876         {
01877
01878     double pivot = arr[high];
01879     int i = (low - 1);
01880
01881     for (int j = low; j <= high - 1; j++)
01882         if (arr[j] <= pivot) {
01883             i++;
01884             SWAP(arr[i], arr[j], double);
01885             SWAP(brr[i], brr[j], int);
01886         }
01887     SWAP(arr[high], arr[i + 1], double);
01888     SWAP(brr[high], brr[i + 1], int);
01889
01890     return (i + 1);
01891 }
```

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

Read atmospheric data.

Definition at line 1895 of file libtrac.c.

```

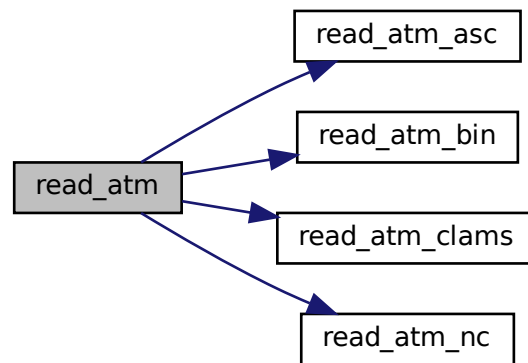
01898         {
01899
01900     int result;
01901
01902     /* Set timer... */
01903     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
01904
01905     /* Init... */
01906     atm->np = 0;
01907
01908     /* Write info... */
01909     LOG(1, "Read atmospheric data: %s", filename);
01910
01911     /* Read ASCII data... */
01912     if (ctl->atm_type == 0)
01913         result = read_atm_asc(filename, ctl, atm);
01914
01915     /* Read binary data... */
01916     else if (ctl->atm_type == 1)
01917         result = read_atm_bin(filename, ctl, atm);
01918
01919     /* Read netCDF data... */
01920     else if (ctl->atm_type == 2)
01921         result = read_atm_nc(filename, ctl, atm);
01922
01923     /* Read CLaMS data... */
01924     else if (ctl->atm_type == 3)
01925         result = read_atm_clams(filename, ctl, atm);
01926
01927     /* Error... */
01928     else
01929         ERRMSG("Atmospheric data type not supported!");
01930
01931     /* Check result... */
01932     if (result != 1)
01933         return 0;
01934
01935     /* Check number of air parcels... */
01936     if (atm->np < 1)
01937         ERRMSG("Can not read any data!");
01938
01939     /* Write info... */
01940     double mini, maxi;
01941     LOG(2, "Number of particles: %d", atm->np);
01942     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
01943     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
01944     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
01945     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
01946     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
01947     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
01948     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
01949     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
```

```

01950 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
01951 for (int iq = 0; iq < ctl->nq; iq++) {
01952     char msg[LEN];
01953     sprintf(msg, "Quantity %s range: %s ... %s %s",
01954             ctl->qnt_name[iq], ctl->qnt_format[iq],
01955             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
01956     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
01957     LOG(2, msg, mini, maxi);
01958 }
01959
01960 /* Return success... */
01961 return 1;
01962 }

```

Here is the call graph for this function:



**5.19.2.31 read\_atm\_asc()** int read\_atm\_asc (

```

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

```

Read atmospheric data in ASCII format.

Definition at line 1966 of file `libtrac.c`.

```

01969     {
01970
01971     FILE *in;
01972
01973     /* Open file... */
01974     if (!(in = fopen(filename, "r"))) {
01975         WARN("Cannot open file!");
01976         return 0;
01977     }
01978
01979     /* Read line... */
01980     char line[LEN];
01981     while (fgets(line, LEN, in)) {
01982
01983         /* Read data... */
01984         char *tok;
01985         TOK(line, tok, "%lg", atm->time[atm->np]);
01986         TOK(NULL, tok, "%lg", atm->p[atm->np]);
01987         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
01988         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
01989         for (int iq = 0; iq < ctl->nq; iq++)
01990             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

```



```

01991
01992     /* Convert altitude to pressure... */
01993     atm->p[atm->np] = P(atm->p[atm->np]);
01994
01995     /* Increment data point counter... */
01996     if ((++atm->np) > NP)
01997         ERRMSG("Too many data points!");
01998 }
01999
02000 /* Close file... */
02001 fclose(in);
02002
02003 /* Return success... */
02004 return 1;
02005 }

```

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

Read atmospheric data in binary format.

Definition at line 2009 of file libtrac.c.

```

02012     {
02013
02014     FILE *in;
02015
02016     /* Open file... */
02017     if (!(in = fopen(filename, "r")))
02018         return 0;
02019
02020     /* Check version of binary data... */
02021     int version;
02022     FREAD(&version, int,
02023          1,
02024          in);
02025     if (version != 100)
02026         ERRMSG("Wrong version of binary data!");
02027
02028     /* Read data... */
02029     FREAD(&atm->np, int,
02030          1,
02031          in);
02032     FREAD(atm->time, double,
02033          (size_t) atm->np,
02034          in);
02035     FREAD(atm->p, double,
02036          (size_t) atm->np,
02037          in);
02038     FREAD(atm->lon, double,
02039          (size_t) atm->np,
02040          in);
02041     FREAD(atm->lat, double,
02042          (size_t) atm->np,
02043          in);
02044     for (int iq = 0; iq < ctl->nq; iq++)
02045         FREAD(atm->q[iq], double,
02046              (size_t) atm->np,
02047              in);
02048
02049     /* Read final flag... */
02050     int final;
02051     FREAD(&final, int,
02052          1,
02053          in);
02054     if (final != 999)
02055         ERRMSG("Error while reading binary data!");
02056
02057     /* Close file... */
02058     fclose(in);
02059
02060     /* Return success... */
02061     return 1;
02062 }

```

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

Read atmospheric data in CLaMS format.

Definition at line 2066 of file libtrac.c.

```
02069     {
02070
02071     int ncid, varid;
02072
02073     /* Open file... */
02074     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02075         return 0;
02076
02077     /* Get dimensions... */
02078     NC_INQ_DIM("NPARTS", &atm->np, 1, NP);
02079
02080     /* Get time... */
02081     if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
02082         NC(nc_get_var_double(ncid, varid, atm->time));
02083     } else {
02084         WARN("TIME_INIT not found use time instead!");
02085         double time_init;
02086         NC_GET_DOUBLE("time", &time_init, 1);
02087         for (int ip = 0; ip < atm->np; ip++) {
02088             atm->time[ip] = time_init;
02089         }
02090     }
02091
02092     /* Read zeta coordinate, pressure is optional... */
02093     if (ctl->vert_coord_ap == 1) {
02094         NC_GET_DOUBLE("ZETA", atm->zeta, 1);
02095         NC_GET_DOUBLE("PRESS", atm->p, 0);
02096     }
02097
02098     /* Read pressure, zeta coordinate is optional... */
02099     else {
02100         NC_GET_DOUBLE("PRESS", atm->p, 1);
02101         NC_GET_DOUBLE("ZETA", atm->zeta, 0);
02102     }
02103
02104     /* Read longitude and latitude... */
02105     NC_GET_DOUBLE("LON", atm->lon, 1);
02106     NC_GET_DOUBLE("LAT", atm->lat, 1);
02107
02108     /* Close file... */
02109     NC(nc_close(ncid));
02110
02111     /* Return success... */
02112     return 1;
02113 }
```

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

Read atmospheric data in netCDF format.

Definition at line 2117 of file libtrac.c.

```
02120     {
02121
02122     int ncid, varid;
02123
02124     /* Open file... */
02125     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02126         return 0;
02127
02128     /* Get dimensions... */
02129     NC_INQ_DIM("obs", &atm->np, 1, NP);
02130
02131     /* Read geolocations... */
02132     NC_GET_DOUBLE("time", atm->time, 1);
```

```

02133 NC_GET_DOUBLE("press", atm->p, 1);
02134 NC_GET_DOUBLE("lon", atm->lon, 1);
02135 NC_GET_DOUBLE("lat", atm->lat, 1);
02136
02137 /* Read variables... */
02138 for (int iq = 0; iq < ctl->nq; iq++)
02139     NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
02140
02141 /* Close file... */
02142 NC(nc_close(ncid));
02143
02144 /* Return success... */
02145 return 1;
02146 }

```

**5.19.2.35 read\_clim()** void read\_clim (  
     ctl\_t \* ctl,  
     clim\_t \* clim )

Read climatological data.

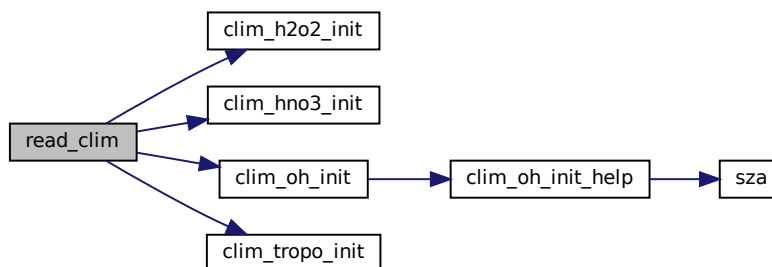
Definition at line 2150 of file libtrac.c.

```

02152 {
02153
02154     /* Set timer... */
02155     SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);
02156
02157     /* Init tropopause climatology... */
02158     clim_tropo_init(clim);
02159
02160     /* Init HNO3 climatology... */
02161     clim_hno3_init(clim);
02162
02163     /* Read OH climatology... */
02164     if (ctl->clim_oh_filename[0] != '-')
02165         clim_oh_init(ctl, clim);
02166
02167     /* Read H2O2 climatology... */
02168     if (ctl->clim_h2o2_filename[0] != '-')
02169         clim_h2o2_init(ctl, clim);
02170 }

```

Here is the call graph for this function:



```

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

```

Read control parameters.

Definition at line 2174 of file [libtrac.c](#).

```

2178     {
2179
2180     /* Set timer... */
2181     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
2182
2183     /* Write info... */
2184     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
2185         "(executable: %s | version: %s | compiled: %s, %s)\n",
2186         argv[0], VERSION, __DATE__, __TIME__);
2187
2188     /* Initialize quantity indices... */
2189     ctl->qnt_idx = -1;
2190     ctl->qnt_ens = -1;
2191     ctl->qnt_stat = -1;
2192     ctl->qnt_m = -1;
2193     ctl->qnt_vmr = -1;
2194     ctl->qnt_rp = -1;
2195     ctl->qnt_rhop = -1;
2196     ctl->qnt_ps = -1;
2197     ctl->qnt_ts = -1;
2198     ctl->qnt_zs = -1;
2199     ctl->qnt_us = -1;
2200     ctl->qnt_vs = -1;
2201     ctl->qnt_pbl = -1;
2202     ctl->qnt_pt = -1;
2203     ctl->qnt_tt = -1;
2204     ctl->qnt_zt = -1;
2205     ctl->qnt_h2ot = -1;
2206     ctl->qnt_z = -1;
2207     ctl->qnt_p = -1;
2208     ctl->qnt_t = -1;
2209     ctl->qnt_rho = -1;
2210     ctl->qnt_u = -1;
2211     ctl->qnt_v = -1;
2212     ctl->qnt_w = -1;
2213     ctl->qnt_h2o = -1;
2214     ctl->qnt_o3 = -1;
2215     ctl->qnt_lwc = -1;
2216     ctl->qnt_iwc = -1;
2217     ctl->qnt_pct = -1;
2218     ctl->qnt_pcb = -1;
2219     ctl->qnt_cl = -1;
2220     ctl->qnt_plcl = -1;
2221     ctl->qnt_plfc = -1;
2222     ctl->qnt_pel = -1;
2223     ctl->qnt_cape = -1;
2224     ctl->qnt_cin = -1;
2225     ctl->qnt_hno3 = -1;
2226     ctl->qnt_oh = -1;
2227     ctl->qnt_vmrimpl = -1;
2228     ctl->qnt_mloss_oh = -1;
2229     ctl->qnt_mloss_h2o2 = -1;
2230     ctl->qnt_mloss_wet = -1;
2231     ctl->qnt_mloss_dry = -1;
2232     ctl->qnt_mloss_decay = -1;
2233     ctl->qnt_psat = -1;
2234     ctl->qnt_psice = -1;
2235     ctl->qnt_pw = -1;
2236     ctl->qnt_sh = -1;
2237     ctl->qnt_rh = -1;
2238     ctl->qnt_rhice = -1;
2239     ctl->qnt_theta = -1;
2240     ctl->qnt_zeta = -1;
2241     ctl->qnt_tvirt = -1;
2242     ctl->qnt_lapse = -1;
2243     ctl->qnt_vh = -1;
2244     ctl->qnt_vz = -1;
2245     ctl->qnt_pv = -1;
2246     ctl->qnt_tdew = -1;
2247     ctl->qnt_tice = -1;
2248     ctl->qnt_tsts = -1;
2249     ctl->qnt_tnat = -1;
2250
2251     /* Read quantities... */

```

```

02252     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02253     if (ctl->nq > NQ)
02254         ERRMSG("Too many quantities!");
02255     for (int iq = 0; iq < ctl->nq; iq++) {
02256
02257         /* Read quantity name and format... */
02258         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02259         scan_ctl(filename, argc, argv, "QNT_LONGNAME", iq, ctl->qnt_name[iq],
02260             ctl->qnt_longname[iq]);
02261         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02262             ctl->qnt_format[iq]);
02263
02264         /* Try to identify quantity... */
02265         SET_QNT(qnt_idx, "idx", "particle index", "-")
02266         SET_QNT(qnt_ens, "ens", "ensemble index", "-")
02267         SET_QNT(qnt_stat, "stat", "station flag", "-")
02268         SET_QNT(qnt_m, "m", "mass", "kg")
02269         SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
02270         SET_QNT(qnt_rp, "rp", "particle radius", "microns")
02271         SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
02272         SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
02273         SET_QNT(qnt_ts, "ts", "surface temperature", "K")
02274         SET_QNT(qnt_zs, "zs", "surface height", "km")
02275         SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
02276         SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
02277         SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
02278         SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
02279         SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
02280         SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
02281         SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
02282         SET_QNT(qnt_z, "z", "geopotential height", "km")
02283         SET_QNT(qnt_p, "p", "pressure", "hPa")
02284         SET_QNT(qnt_t, "t", "temperature", "K")
02285         SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
02286         SET_QNT(qnt_u, "u", "zonal wind", "m/s")
02287         SET_QNT(qnt_v, "v", "meridional wind", "m/s")
02288         SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
02289         SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
02290         SET_QNT(qnt_o3, "o3", "ozone", "ppv")
02291         SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
02292         SET_QNT(qnt_lwc, "lwc", "cloud liquid water content", "kg/kg")
02293         SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
02294         SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
02295         SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
02296         SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
02297         SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
02298         SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
02299         SET_QNT(qnt_cape, "cape", "convective available potential energy",
02300             "J/kg")
02301         SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
02302         SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
02303         SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
02304         SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
02305         SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
02306         SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
02307             "kg")
02308         SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
02309             "kg")
02310         SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
02311             "kg")
02312         SET_QNT(qnt_mloss_decay, "mloss_decay",
02313             "mass loss due to exponential decay", "kg")
02314         SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
02315         SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
02316         SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
02317         SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
02318         SET_QNT(qnt_rh, "rh", "relative humidity", "%")
02319         SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%")
02320         SET_QNT(qnt_theta, "theta", "potential temperature", "K")
02321         SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
02322         SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
02323         SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
02324         SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
02325         SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
02326         SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
02327         SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
02328         SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
02329         SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
02330         SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
02331         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02332     }
02333
02334     /* netCDF I/O parameters... */
02335     ctl->chunkszhint =
02336         (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02337             NULL);
02338     ctl->read_mode =

```

```

02339     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02340
02341     /* Vertical coordinates and velocities... */
02342     ctl->vert_coord_ap =
02343         (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
02344     ctl->vert_coord_met =
02345         (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
02346     ctl->vert_vel =
02347         (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
02348     ctl->clams_met_data =
02349         (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);
02350
02351     /* Time steps of simulation... */
02352     ctl->direction =
02353         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02354     if (ctl->direction != -1 && ctl->direction != 1)
02355         ERRMSG("Set DIRECTION to -1 or 1!");
02356     ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02357     ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02358
02359     /* Meteo data... */
02360     scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02361     ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02362     ctl->met_type =
02363         (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
02364     ctl->met_nc_scale =
02365         (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", NULL);
02366     ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02367     ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02368     ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02369     if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02370         ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02371     ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02372     ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02373     ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02374     if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02375         ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02376     ctl->met_detrend =
02377         scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02378     ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02379     if (ctl->met_np > EP)
02380         ERRMSG("Too many levels!");
02381     for (int ip = 0; ip < ctl->met_np; ip++)
02382         ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02383     ctl->met_geopot_sx =
02384         (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02385     ctl->met_geopot_sy =
02386         (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02387     ctl->met_tropo =
02388         (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02389     if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02390         ERRMSG("Set MET_TROPO = 0 ... 5!");
02391     ctl->met_tropo_lapse =
02392         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02393     ctl->met_tropo_nlev =
02394         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02395     ctl->met_tropo_lapse_sep =
02396         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02397     ctl->met_tropo_nlev_sep =
02398         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02399             NULL);
02400     ctl->met_tropo_pv =
02401         scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02402     ctl->met_tropo_theta =
02403         scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02404     ctl->met_tropo_spline =
02405         (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02406     ctl->met_cloud =
02407         (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02408     if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02409         ERRMSG("Set MET_CLOUD = 0 ... 3!");
02410     ctl->met_cloud_min =
02411         scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
02412     ctl->met_dt_out =
02413         scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02414     ctl->met_cache =
02415         (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02416
02417     /* Sorting... */
02418     ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);
02419
02420     /* Isosurface parameters... */
02421     ctl->isosurf =
02422         (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02423     scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02424
02425     /* Advection parameters... */

```

```

02426     ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
02427     if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
02428         ERRMSG("Set ADVECT to 1, 2, or 4!");
02429     ctl->reflect =
02430         (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02431
02432     /* Diffusion parameters... */
02433     ctl->turb_dx_trop =
02434         scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02435     ctl->turb_dx_strat =
02436         scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02437     ctl->turb_dz_trop =
02438         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02439     ctl->turb_dz_strat =
02440         scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02441     ctl->turb_mesox =
02442         scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02443     ctl->turb_mesoz =
02444         scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02445
02446     /* Convection... */
02447     ctl->conv_cape =
02448         scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02449     ctl->conv_cin =
02450         scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02451     ctl->conv_wmax =
02452         scan_ctl(filename, argc, argv, "CONV_WMAX", -1, "-999", NULL);
02453     ctl->conv_wcape =
02454         (int) scan_ctl(filename, argc, argv, "CONV_WCAPE", -1, "0", NULL);
02455     ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02456     ctl->conv_mix =
02457         (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "0", NULL);
02458     ctl->conv_mix_bot =
02459         (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02460     ctl->conv_mix_top =
02461         (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02462
02463     /* Boundary conditions... */
02464     ctl->bound_mass =
02465         scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02466     ctl->bound_mass_trend =
02467         scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
02468     ctl->bound_vmr =
02469         scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02470     ctl->bound_vmr_trend =
02471         scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
02472     ctl->bound_lat0 =
02473         scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02474     ctl->bound_lat1 =
02475         scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02476     ctl->bound_p0 =
02477         scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02478     ctl->bound_p1 =
02479         scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02480     ctl->bound_dps =
02481         scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02482
02483     /* Species parameters... */
02484     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02485     if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02486         ctl->molmass = 120.907;
02487         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
02488         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
02489     } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02490         ctl->molmass = 137.359;
02491         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
02492         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
02493     } else if (strcasecmp(ctl->species, "CH4") == 0) {
02494         ctl->molmass = 16.043;
02495         ctl->oh_chem_reaction = 2;
02496         ctl->oh_chem[0] = 2.45e-12;
02497         ctl->oh_chem[1] = 1775;
02498         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
02499         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02500     } else if (strcasecmp(ctl->species, "CO") == 0) {
02501         ctl->molmass = 28.01;
02502         ctl->oh_chem_reaction = 3;
02503         ctl->oh_chem[0] = 6.9e-33;
02504         ctl->oh_chem[1] = 2.1;
02505         ctl->oh_chem[2] = 1.1e-12;
02506         ctl->oh_chem[3] = -1.3;
02507         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
02508         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
02509     } else if (strcasecmp(ctl->species, "CO2") == 0) {
02510         ctl->molmass = 44.009;
02511         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
02512         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;

```

```

02513 } else if (strcasecmp(ctl->species, "H2O") == 0) {
02514     ctl->molmass = 18.01528;
02515 } else if (strcasecmp(ctl->species, "N2O") == 0) {
02516     ctl->molmass = 44.013;
02517     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
02518     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
02519 } else if (strcasecmp(ctl->species, "NH3") == 0) {
02520     ctl->molmass = 17.031;
02521     ctl->oh_chem_reaction = 2;
02522     ctl->oh_chem[0] = 1.7e-12;
02523     ctl->oh_chem[1] = 710;
02524     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
02525     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
02526 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02527     ctl->molmass = 63.012;
02528     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
02529     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
02530 } else if (strcasecmp(ctl->species, "NO") == 0) {
02531     ctl->molmass = 30.006;
02532     ctl->oh_chem_reaction = 3;
02533     ctl->oh_chem[0] = 7.1e-31;
02534     ctl->oh_chem[1] = 2.6;
02535     ctl->oh_chem[2] = 3.6e-11;
02536     ctl->oh_chem[3] = 0.1;
02537     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
02538     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02539 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02540     ctl->molmass = 46.005;
02541     ctl->oh_chem_reaction = 3;
02542     ctl->oh_chem[0] = 1.8e-30;
02543     ctl->oh_chem[1] = 3.0;
02544     ctl->oh_chem[2] = 2.8e-11;
02545     ctl->oh_chem[3] = 0.0;
02546     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
02547     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02548 } else if (strcasecmp(ctl->species, "O3") == 0) {
02549     ctl->molmass = 47.997;
02550     ctl->oh_chem_reaction = 2;
02551     ctl->oh_chem[0] = 1.7e-12;
02552     ctl->oh_chem[1] = 940;
02553     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
02554     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
02555 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02556     ctl->molmass = 146.048;
02557     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
02558     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
02559 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02560     ctl->molmass = 64.066;
02561     ctl->oh_chem_reaction = 3;
02562     ctl->oh_chem[0] = 2.9e-31;
02563     ctl->oh_chem[1] = 4.1;
02564     ctl->oh_chem[2] = 1.7e-12;
02565     ctl->oh_chem[3] = -0.2;
02566     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
02567     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
02568 } else {
02569     ctl->molmass =
02570         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02571     ctl->oh_chem_reaction =
02572         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02573     ctl->h2o2_chem_reaction =
02574         (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
02575             NULL);
02576     for (int ip = 0; ip < 4; ip++)
02577         ctl->oh_chem[ip] =
02578             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02579     for (int ip = 0; ip < 1; ip++)
02580         ctl->dry_depo[ip] =
02581             scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02582     ctl->wet_depo_ic_a =
02583         scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
02584     ctl->wet_depo_ic_b =
02585         scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
02586     ctl->wet_depo_bc_a =
02587         scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
02588     ctl->wet_depo_bc_b =
02589         scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
02590     for (int ip = 0; ip < 3; ip++)
02591         ctl->wet_depo_ic_h[ip] =
02592             scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
02593     for (int ip = 0; ip < 1; ip++)
02594         ctl->wet_depo_bc_h[ip] =
02595             scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
02596 }
02597
02598 /* Wet deposition... */
02599 ctl->wet_depo_pre[0] =

```



```

02600     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
02601     ctl->wet_depo_pre[1] =
02602     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
02603     ctl->wet_depo_ic_ret_ratio =
02604     scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
02605     ctl->wet_depo_bc_ret_ratio =
02606     scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);
02607
02608     /* OH chemistry... */
02609     ctl->oh_chem_beta =
02610     scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
02611     scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
02612     "../data/clams_radical_species.nc", ctl->clim_oh_filename);
02613
02614     /* H2O2 chemistry... */
02615     ctl->h2o2_chem_cc =
02616     scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
02617     scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
02618     "../data/cams_H2O2.nc", ctl->clim_h2o2_filename);
02619
02620     /* Exponential decay... */
02621     ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02622     ctl->tdec_strat =
02623     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02624
02625     /* PSC analysis... */
02626     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02627     ctl->psc_hno3 =
02628     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02629
02630     /* Output of atmospheric data... */
02631     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02632     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02633     ctl->atm_dt_out =
02634     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02635     ctl->atm_filter =
02636     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02637     ctl->atm_stride =
02638     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02639     ctl->atm_type =
02640     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02641
02642     /* Output of CSI data... */
02643     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02644     ctl->csi_dt_out =
02645     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02646     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02647     ctl->csi_obsmin =
02648     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02649     ctl->csi_modmin =
02650     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02651     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02652     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02653     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02654     ctl->csi_lon0 =
02655     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02656     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02657     ctl->csi_nx =
02658     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02659     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02660     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02661     ctl->csi_ny =
02662     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02663
02664     /* Output of ensemble data... */
02665     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02666     ctl->ens_dt_out =
02667     scan_ctl(filename, argc, argv, "ENS_DT_OUT", -1, "86400", NULL);
02668
02669     /* Output of grid data... */
02670     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02671     ctl->grid_basename);
02672     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02673     ctl->grid_dt_out =
02674     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02675     ctl->grid_sparse =
02676     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02677     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02678     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02679     ctl->grid_nz =
02680     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02681     ctl->grid_lon0 =
02682     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02683     ctl->grid_lon1 =
02684     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02685     ctl->grid_nx =
02686     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);

```

```

02687   ctl->grid_lat0 =
02688       scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02689   ctl->grid_lat1 =
02690       scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02691   ctl->grid_ny =
02692       (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02693   ctl->grid_type =
02694       (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", NULL);
02695
02696   /* Output of profile data... */
02697   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02698       ctl->prof_basename);
02699   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02700   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02701   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02702   ctl->prof_nz =
02703       (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02704   ctl->prof_lon0 =
02705       scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02706   ctl->prof_lon1 =
02707       scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02708   ctl->prof_nx =
02709       (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02710   ctl->prof_lat0 =
02711       scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02712   ctl->prof_lat1 =
02713       scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02714   ctl->prof_ny =
02715       (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02716
02717   /* Output of sample data... */
02718   scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02719       ctl->sample_basename);
02720   scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02721       ctl->sample_obsfile);
02722   ctl->sample_dx =
02723       scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02724   ctl->sample_dz =
02725       scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02726
02727   /* Output of station data... */
02728   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02729       ctl->stat_basename);
02730   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02731   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02732   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02733   ctl->stat_t0 =
02734       scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02735   ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02736 }

```

Here is the call graph for this function:



**5.19.2.37 read\_met()** int read\_met (

```

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

```

Read meteo data file.

Definition at line 2740 of file libtrac.c.

```

02744     {
02745
02746     /* Write info... */
02747     LOG(1, "Read meteo data: %s", filename);
02748
02749     /* Read netCDF data... */
02750     if (ctl->met_type == 0) {
02751
02752         int ncid;
02753
02754         /* Open netCDF file... */
02755         if (nc_open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02756             NC_NOERR) {
02757             WARN("Cannot open file!");
02758             return 0;
02759         }
02760
02761         /* Read coordinates of meteo data... */
02762         read_met_grid(filename, ncid, ctl, met);
02763
02764         /* Read meteo data on vertical levels... */
02765         read_met_levels(ncid, ctl, met);
02766
02767         /* Extrapolate data for lower boundary... */
02768         read_met_extrapolate(met);
02769
02770         /* Read surface data... */
02771         read_met_surface(ncid, met, ctl);
02772
02773         /* Create periodic boundary conditions... */
02774         read_met_periodic(met);
02775
02776         /* Downsampling... */
02777         read_met_sample(ctl, met);
02778
02779         /* Calculate geopotential heights... */
02780         read_met_geopot(ctl, met);
02781
02782         /* Calculate potential vorticity... */
02783         read_met_pv(met);
02784
02785         /* Calculate boundary layer data... */
02786         read_met_pbl(met);
02787
02788         /* Calculate tropopause data... */
02789         read_met_tropo(ctl, clim, met);
02790
02791         /* Calculate cloud properties... */
02792         read_met_cloud(ctl, met);
02793
02794         /* Calculate convective available potential energy... */
02795         read_met_cape(clim, met);
02796
02797         /* Detrending... */
02798         read_met_detrend(ctl, met);
02799
02800         /* Close file... */
02801         NC(nc_close(ncid));
02802     }
02803
02804     /* Read binary data... */
02805     else if (ctl->met_type >= 1 && ctl->met_type <= 4) {
02806
02807         FILE *in;
02808
02809         double r;
02810
02811         int year, mon, day, hour, min, sec;
02812
02813         /* Set timer... */
02814         SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);
02815
02816         /* Open file... */
02817         if (!(in = fopen(filename, "r"))) {
02818             WARN("Cannot open file!");
02819             return 0;
02820         }
02821
02822         /* Check type of binary data... */
02823         int met_type;
02824         FREAD(&met_type, int,
02825             1,
02826             in);
02827         if (met_type != ctl->met_type)
02828             ERRMSG("Wrong MET_TYPE of binary data!");
02829

```

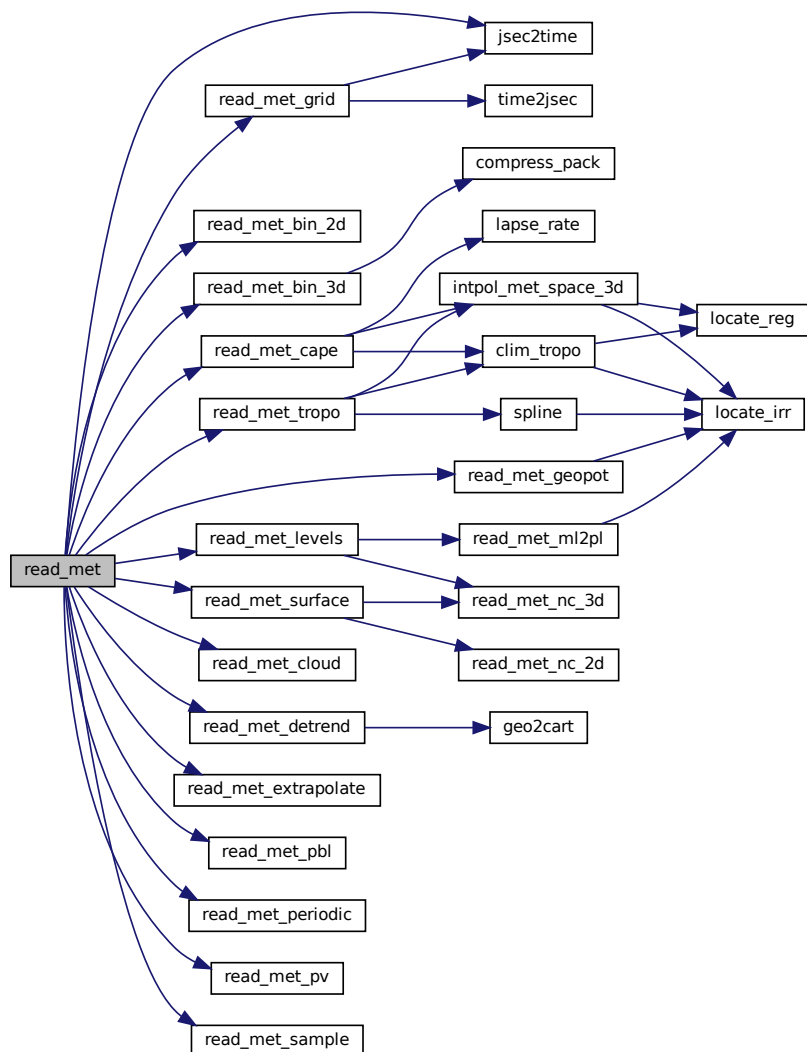
```

02830      /* Check version of binary data... */
02831      int version;
02832      FREAD(&version, int,
02833           1,
02834           in);
02835      if (version != 100)
02836          ERRMSG("Wrong version of binary data!");
02837
02838      /* Read time... */
02839      FREAD(&met->time, double,
02840           1,
02841           in);
02842      jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
02843      LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
02844          met->time, year, mon, day, hour, min);
02845      if (year < 1900 || year > 2100 || mon < 1 || mon > 12
02846          || day < 1 || day > 31 || hour < 0 || hour > 23)
02847          ERRMSG("Error while reading time!");
02848
02849      /* Read dimensions... */
02850      FREAD(&met->nx, int,
02851           1,
02852           in);
02853      LOG(2, "Number of longitudes: %d", met->nx);
02854      if (met->nx < 2 || met->nx > EX)
02855          ERRMSG("Number of longitudes out of range!");
02856
02857      FREAD(&met->ny, int,
02858           1,
02859           in);
02860      LOG(2, "Number of latitudes: %d", met->ny);
02861      if (met->ny < 2 || met->ny > EY)
02862          ERRMSG("Number of latitudes out of range!");
02863
02864      FREAD(&met->np, int,
02865           1,
02866           in);
02867      LOG(2, "Number of levels: %d", met->np);
02868      if (met->np < 2 || met->np > EP)
02869          ERRMSG("Number of levels out of range!");
02870
02871      /* Read grid... */
02872      FREAD(met->lon, double,
02873           (size_t) met->nx,
02874           in);
02875      LOG(2, "Longitudes: %g, %g ... %g deg",
02876          met->lon[0], met->lon[1], met->lon[met->nx - 1]);
02877
02878      FREAD(met->lat, double,
02879           (size_t) met->ny,
02880           in);
02881      LOG(2, "Latitudes: %g, %g ... %g deg",
02882          met->lat[0], met->lat[1], met->lat[met->ny - 1]);
02883
02884      FREAD(met->p, double,
02885           (size_t) met->np,
02886           in);
02887      LOG(2, "Altitude levels: %g, %g ... %g km",
02888          Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
02889      LOG(2, "Pressure levels: %g, %g ... %g hPa",
02890          met->p[0], met->p[1], met->p[met->np - 1]);
02891
02892      /* Read surface data... */
02893      read_met_bin_2d(in, met, met->ps, "PS");
02894      read_met_bin_2d(in, met, met->ts, "TS");
02895      read_met_bin_2d(in, met, met->zs, "ZS");
02896      read_met_bin_2d(in, met, met->us, "US");
02897      read_met_bin_2d(in, met, met->vs, "VS");
02898      read_met_bin_2d(in, met, met->pbl, "PBL");
02899      read_met_bin_2d(in, met, met->pt, "PT");
02900      read_met_bin_2d(in, met, met->tt, "TT");
02901      read_met_bin_2d(in, met, met->zt, "ZT");
02902      read_met_bin_2d(in, met, met->h2ot, "H2OT");
02903      read_met_bin_2d(in, met, met->pct, "PCT");
02904      read_met_bin_2d(in, met, met->pcb, "PCB");
02905      read_met_bin_2d(in, met, met->cl, "CL");
02906      read_met_bin_2d(in, met, met->plcl, "PLCL");
02907      read_met_bin_2d(in, met, met->plfc, "PLFC");
02908      read_met_bin_2d(in, met, met->pel, "PEL");
02909      read_met_bin_2d(in, met, met->cape, "CAPE");
02910      read_met_bin_2d(in, met, met->cin, "CIN");
02911
02912      /* Read level data... */
02913      read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
02914      read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
02915      read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
02916      read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);

```

```
02917     read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
02918     read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
02919     read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
02920     read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
02921     read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
02922     read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);
02923
02924     /* Read final flag... */
02925     int final;
02926     FREAD(&final, int,
02927          1,
02928          in);
02929     if (final != 999)
02930         ERRMSG("Error while reading binary data!");
02931
02932     /* Close file... */
02933     fclose(in);
02934 }
02935
02936 /* Not implemented... */
02937 else
02938     ERRMSG("MET_TYPE not implemented!");
02939
02940 /* Copy wind data to cache... */
02941 #ifdef UVW
02942 #pragma omp parallel for default(shared) collapse(2)
02943     for (int ix = 0; ix < met->nx; ix++)
02944         for (int iy = 0; iy < met->ny; iy++)
02945             for (int ip = 0; ip < met->np; ip++) {
02946                 met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
02947                 met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
02948                 met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
02949             }
02950 #endif
02951
02952 /* Return success... */
02953 return 1;
02954 }
```

Here is the call graph for this function:



**5.19.2.38 read\_met\_bin\_2d()** void read\_met\_bin\_2d (  
FILE \* in,  
met\_t \* met,  
float var[EX][EY],  
char \* varname )

Read 2-D meteo variable.

Definition at line 2958 of file libtrac.c.

```

02962     {
02963
02964     float *help;
02965
02966     /* Allocate... */
02967     ALLOC(help, float,
02968           EX * EY);

```

```

02969
02970 /* Read uncompressed... */
02971 LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
02972 FREAD(help, float,
02973       (size_t) (met->nx * met->ny),
02974       in);
02975
02976 /* Copy data... */
02977 for (int ix = 0; ix < met->nx; ix++)
02978     for (int iy = 0; iy < met->ny; iy++)
02979         var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];
02980
02981 /* Free... */
02982 free(help);
02983 }

```

### 5.19.2.39 read\_met\_bin\_3d() void read\_met\_bin\_3d (

```

FILE * in,
ctl_t * ctl,
met_t * met,
float var[EX][EY][EP],
char * varname,
int precision,
double tolerance )

```

Read 3-D meteo variable.

Definition at line 2987 of file libtrac.c.

```

02994 {
02995
02996     float *help;
02997
02998     /* Allocate... */
02999     ALLOC(help, float,
03000           EX * EY * EP);
03001
03002     /* Read uncompressed data... */
03003     if (ctl->met_type == 1) {
03004         LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
03005         FREAD(help, float,
03006               (size_t) (met->nx * met->ny * met->np),
03007               in);
03008     }
03009
03010     /* Read packed data... */
03011     else if (ctl->met_type == 2)
03012         compress_pack(varname, help, (size_t) (met->ny * met->nx),
03013                       (size_t) met->np, 1, in);
03014
03015     /* Read zfp data... */
03016     else if (ctl->met_type == 3) {
03017 #ifdef ZFP
03018         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
03019                     tolerance, 1, in);
03020 #else
03021         ERRMSG("zfp compression not supported!");
03022         LOG(3, "%d %g", precision, tolerance);
03023 #endif
03024     }
03025
03026     /* Read zstd data... */
03027     else if (ctl->met_type == 4) {
03028 #ifdef ZSTD
03029         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
03030                      in);
03031 #else
03032         ERRMSG("zstd compression not supported!");
03033 #endif
03034     }
03035
03036     /* Copy data... */
03037 #pragma omp parallel for default(shared) collapse(2)
03038     for (int ix = 0; ix < met->nx; ix++)
03039         for (int iy = 0; iy < met->ny; iy++)
03040             for (int ip = 0; ip < met->np; ip++)
03041                 var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];

```

```

03042
03043  /* Free... */
03044  free(help);
03045 }

```

Here is the call graph for this function:



**5.19.2.40 read\_met\_cape()** void read\_met\_cape (  
     clim\_t \* clim,  
     met\_t \* met )

Calculate convective available potential energy.

Definition at line 3049 of file libtrac.c.

```

03051     {
03052
03053     /* Set timer... */
03054     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
03055     LOG(2, "Calculate CAPE...");
03056
03057     /* Vertical spacing (about 100 m)... */
03058     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
03059
03060     /* Loop over columns... */
03061     #pragma omp parallel for default(shared) collapse(2)
03062     for (int ix = 0; ix < met->nx; ix++)
03063         for (int iy = 0; iy < met->ny; iy++) {
03064
03065             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
03066             int n = 0;
03067             double h2o = 0, t, theta = 0;
03068             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
03069             double ptop = pbot - 50.;
03070             for (int ip = 0; ip < met->np; ip++) {
03071                 if (met->p[ip] <= pbot) {
03072                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
03073                     h2o += met->h2o[ix][iy][ip];
03074                     n++;
03075                 }
03076                 if (met->p[ip] < ptop && n > 0)
03077                     break;
03078             }
03079             theta /= n;
03080             h2o /= n;
03081
03082             /* Cannot compute anything if water vapor is missing... */
03083             met->plcl[ix][iy] = GSL_NAN;
03084             met->plfc[ix][iy] = GSL_NAN;
03085             met->pel[ix][iy] = GSL_NAN;
03086             met->cape[ix][iy] = GSL_NAN;
03087             met->cin[ix][iy] = GSL_NAN;
03088             if (h2o <= 0)
03089                 continue;
03090
03091             /* Find lifted condensation level (LCL)... */
03092             ptop = P(20.);
03093             pbot = met->ps[ix][iy];
03094             do {
03095                 met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
03096                 t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
03097                 if (RH(met->plcl[ix][iy], t, h2o) > 100.)

```

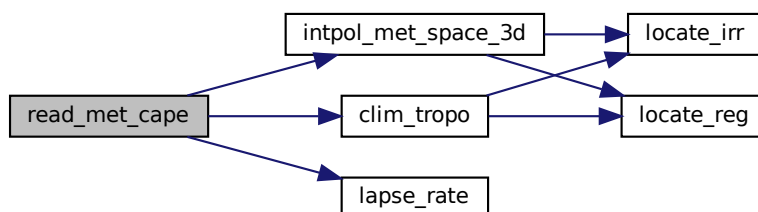


```

03098     ptop = met->plcl[ix][iy];
03099     else
03100     pbot = met->plcl[ix][iy];
03101 } while (pbot - ptop > 0.1);
03102
03103 /* Calculate CIN up to LCL... */
03104 INTPOL_INIT;
03105 double dcape, dz, h2o_env, t_env;
03106 double p = met->ps[ix][iy];
03107 met->cape[ix][iy] = met->cin[ix][iy] = 0;
03108 do {
03109     dz = dz0 * TVIRT(t, h2o);
03110     p /= pfac;
03111     t = theta / pow(1000. / p, 0.286);
03112     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03113         &t_env, ci, cw, 1);
03114     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03115         &h2o_env, ci, cw, 0);
03116     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03117         TVIRT(t_env, h2o_env) * dz;
03118     if (dcape < 0)
03119         met->cin[ix][iy] += fabsf((float) dcape);
03120 } while (p > met->plcl[ix][iy]);
03121
03122 /* Calculate level of free convection (LFC), equilibrium level (EL),
03123    and convective available potential energy (CAPE)... */
03124 dcape = 0;
03125 p = met->plcl[ix][iy];
03126 t = theta / pow(1000. / p, 0.286);
03127 ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
03128 do {
03129     dz = dz0 * TVIRT(t, h2o);
03130     p /= pfac;
03131     t -= lapse_rate(t, h2o) * dz;
03132     double psat = PSAT(t);
03133     h2o = psat / (p - (1. - EPS) * psat);
03134     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03135         &t_env, ci, cw, 1);
03136     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03137         &h2o_env, ci, cw, 0);
03138     double dcape_old = dcape;
03139     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03140         TVIRT(t_env, h2o_env) * dz;
03141     if (dcape > 0) {
03142         met->cape[ix][iy] += (float) dcape;
03143         if (!isfinite(met->plfc[ix][iy]))
03144             met->plfc[ix][iy] = (float) p;
03145     } else if (dcape_old > 0)
03146         met->pel[ix][iy] = (float) p;
03147     if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
03148         met->cin[ix][iy] += fabsf((float) dcape);
03149 } while (p > ptop);
03150
03151 /* Check results... */
03152 if (!isfinite(met->plfc[ix][iy]))
03153     met->cin[ix][iy] = GSL_NAN;
03154 }
03155 }

```

Here is the call graph for this function:



**5.19.2.41 read\_met\_cloud()** void read\_met\_cloud (  
     ctl\_t \* ctl,  
     met\_t \* met )

Calculate cloud properties.

Definition at line 3159 of file libtrac.c.

```

03161     {
03162
03163     /* Set timer... */
03164     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
03165     LOG(2, "Calculate cloud data...");
03166
03167     /* Loop over columns... */
03168     #pragma omp parallel for default(shared) collapse(2)
03169     for (int ix = 0; ix < met->nx; ix++)
03170         for (int iy = 0; iy < met->ny; iy++) {
03171
03172         /* Init... */
03173         met->pct[ix][iy] = GSL_NAN;
03174         met->pcb[ix][iy] = GSL_NAN;
03175         met->cl[ix][iy] = 0;
03176
03177         /* Loop over pressure levels... */
03178         for (int ip = 0; ip < met->np - 1; ip++) {
03179
03180             /* Check pressure... */
03181             if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
03182                 continue;
03183
03184             /* Check ice water and liquid water content... */
03185             if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
03186                 || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {
03187
03188                 /* Get cloud top pressure ... */
03189                 met->pct[ix][iy]
03190                     = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
03191
03192                 /* Get cloud bottom pressure ... */
03193                 if (!isfinite(met->pcb[ix][iy]))
03194                     met->pcb[ix][iy]
03195                         = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
03196             }
03197
03198             /* Get cloud water... */
03199             met->cl[ix][iy] += (float)
03200                 (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
03201                     + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
03202                 * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
03203         }
03204     }
03205 }
```

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

Apply detrending method to temperature and winds.

Definition at line 3209 of file libtrac.c.

```

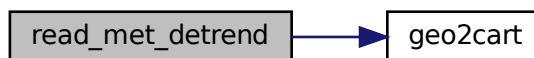
03211     {
03212
03213     met_t *help;
03214
03215     /* Check parameters... */
03216     if (ctl->met_detrend <= 0)
03217         return;
03218
03219     /* Set timer... */
03220     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
03221     LOG(2, "Detrend meteo data...");
03222
03223     /* Allocate... */
03224     ALLOC(help, met_t, 1);
03225
03226     /* Calculate standard deviation... */
```

```

03227 double sigma = ctl->met_detrend / 2.355;
03228 double tssq = 2. * SQR(sigma);
03229
03230 /* Calculate box size in latitude... */
03231 int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03232 sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03233
03234 /* Calculate background... */
03235 #pragma omp parallel for default(shared) collapse(2)
03236 for (int ix = 0; ix < met->nx; ix++) {
03237     for (int iy = 0; iy < met->ny; iy++) {
03238
03239         /* Calculate Cartesian coordinates... */
03240         double x0[3];
03241         geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03242
03243         /* Calculate box size in longitude... */
03244         int sx =
03245             (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03246                 fabs(met->lon[1] - met->lon[0]));
03247         sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03248
03249         /* Init... */
03250         float wsum = 0;
03251         for (int ip = 0; ip < met->np; ip++) {
03252             help->t[ix][iy][ip] = 0;
03253             help->u[ix][iy][ip] = 0;
03254             help->v[ix][iy][ip] = 0;
03255             help->w[ix][iy][ip] = 0;
03256         }
03257
03258         /* Loop over neighboring grid points... */
03259         for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03260             int ix3 = ix2;
03261             if (ix3 < 0)
03262                 ix3 += met->nx;
03263             else if (ix3 >= met->nx)
03264                 ix3 -= met->nx;
03265             for (int iy2 = GSL_MAX(iy - sy, 0);
03266                 iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03267
03268                 /* Calculate Cartesian coordinates... */
03269                 double x1[3];
03270                 geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03271
03272                 /* Calculate weighting factor... */
03273                 float w = (float) exp(-DIST2(x0, x1) / tssq);
03274
03275                 /* Add data... */
03276                 wsum += w;
03277                 for (int ip = 0; ip < met->np; ip++) {
03278                     help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03279                     help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03280                     help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03281                     help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03282                 }
03283             }
03284         }
03285
03286         /* Normalize... */
03287         for (int ip = 0; ip < met->np; ip++) {
03288             help->t[ix][iy][ip] /= wsum;
03289             help->u[ix][iy][ip] /= wsum;
03290             help->v[ix][iy][ip] /= wsum;
03291             help->w[ix][iy][ip] /= wsum;
03292         }
03293     }
03294 }
03295
03296 /* Subtract background... */
03297 #pragma omp parallel for default(shared) collapse(3)
03298 for (int ix = 0; ix < met->nx; ix++)
03299     for (int iy = 0; iy < met->ny; iy++)
03300         for (int ip = 0; ip < met->np; ip++) {
03301             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03302             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03303             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03304             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03305         }
03306
03307 /* Free... */
03308 free(help);
03309 }

```

Here is the call graph for this function:



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

Extrapolate meteo data at lower boundary.

Definition at line 3313 of file libtrac.c.

```

03314     {
03315
03316     /* Set timer... */
03317     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03318     LOG(2, "Extrapolate meteo data...");
03319
03320     /* Loop over columns... */
03321     #pragma omp parallel for default(shared) collapse(2)
03322     for (int ix = 0; ix < met->nx; ix++)
03323     for (int iy = 0; iy < met->ny; iy++) {
03324
03325         /* Find lowest valid data point... */
03326         int ip0;
03327         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03328             if (!isfinite(met->t[ix][iy][ip0])
03329                 || !isfinite(met->u[ix][iy][ip0])
03330                 || !isfinite(met->v[ix][iy][ip0])
03331                 || !isfinite(met->w[ix][iy][ip0]))
03332                 break;
03333
03334         /* Extrapolate... */
03335         for (int ip = ip0; ip >= 0; ip--) {
03336             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03337             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03338             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03339             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03340             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03341             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03342             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03343             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03344         }
03345     }
03346 }
  
```

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

Calculate geopotential heights.

Definition at line 3350 of file libtrac.c.

```

03352     {
03353
03354     static float help[EP][EX][EY];
03355
  
```

```

03356 double logp[EP];
03357
03358 int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03359
03360 /* Set timer... */
03361 SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03362 LOG(2, "Calculate geopotential heights...");
03363
03364 /* Calculate log pressure... */
03365 #pragma omp parallel for default(shared)
03366 for (int ip = 0; ip < met->np; ip++)
03367     logp[ip] = log(met->p[ip]);
03368
03369 /* Apply hydrostatic equation to calculate geopotential heights... */
03370 #pragma omp parallel for default(shared) collapse(2)
03371 for (int ix = 0; ix < met->nx; ix++)
03372     for (int iy = 0; iy < met->ny; iy++) {
03373
03374         /* Get surface height and pressure... */
03375         double zs = met->zs[ix][iy];
03376         double lnps = log(met->ps[ix][iy]);
03377
03378         /* Get temperature and water vapor vmr at the surface... */
03379         int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03380         double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03381             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03382         double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03383             met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03384
03385         /* Upper part of profile... */
03386         met->z[ix][iy][ip0 + 1]
03387             = (float) (zs +
03388                 ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03389                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03390         for (int ip = ip0 + 2; ip < met->np; ip++)
03391             met->z[ix][iy][ip]
03392                 = (float) (met->z[ix][iy][ip - 1] +
03393                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03394                         met->h2o[ix][iy][ip - 1], logp[ip],
03395                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03396
03397         /* Lower part of profile... */
03398         met->z[ix][iy][ip0]
03399             = (float) (zs +
03400                 ZDIFF(lnps, ts, h2os, logp[ip0],
03401                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03402         for (int ip = ip0 - 1; ip >= 0; ip--)
03403             met->z[ix][iy][ip]
03404                 = (float) (met->z[ix][iy][ip + 1] +
03405                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03406                         met->h2o[ix][iy][ip + 1], logp[ip],
03407                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03408     }
03409
03410 /* Check control parameters... */
03411 if (dx == 0 || dy == 0)
03412     return;
03413
03414 /* Default smoothing parameters... */
03415 if (dx < 0 || dy < 0) {
03416     if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03417         dx = 3;
03418         dy = 2;
03419     } else {
03420         dx = 6;
03421         dy = 4;
03422     }
03423 }
03424
03425 /* Calculate weights for smoothing... */
03426 float ws[dx + 1][dy + 1];
03427 #pragma omp parallel for default(shared) collapse(2)
03428 for (int ix = 0; ix <= dx; ix++)
03429     for (int iy = 0; iy < dy; iy++)
03430         ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03431             * (1.0f - (float) iy / (float) dy);
03432
03433 /* Copy data... */
03434 #pragma omp parallel for default(shared) collapse(3)
03435 for (int ix = 0; ix < met->nx; ix++)
03436     for (int iy = 0; iy < met->ny; iy++)
03437         for (int ip = 0; ip < met->np; ip++)
03438             help[ip][ix][iy] = met->z[ix][iy][ip];
03439
03440 /* Horizontal smoothing... */
03441 #pragma omp parallel for default(shared) collapse(3)
03442 for (int ip = 0; ip < met->np; ip++)

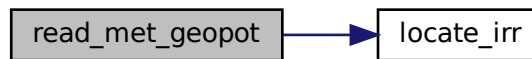
```

```

03443     for (int ix = 0; ix < met->nx; ix++)
03444         for (int iy = 0; iy < met->ny; iy++) {
03445             float res = 0, wsum = 0;
03446             int iy0 = GSL_MAX(iy - dy + 1, 0);
03447             int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03448             for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03449                 int ix3 = ix2;
03450                 if (ix3 < 0)
03451                     ix3 += met->nx;
03452                 else if (ix3 >= met->nx)
03453                     ix3 -= met->nx;
03454                 for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03455                     if (isfinite(help[ip][ix3][iy2])) {
03456                         float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03457                         res += w * help[ip][ix3][iy2];
03458                         wsum += w;
03459                     }
03460             }
03461             if (wsum > 0)
03462                 met->z[ix][iy][ip] = res / wsum;
03463             else
03464                 met->z[ix][iy][ip] = GSL_NAN;
03465         }
03466     }

```

Here is the call graph for this function:



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

```

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

```

Read coordinates of meteo data.

Definition at line 3470 of file libtrac.c.

```

03474     {
03475
03476         char levname[LEN], tstr[10];
03477
03478         double rtime, r2;
03479
03480         int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;
03481
03482         size_t np;
03483
03484         /* Set timer... */
03485         SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03486         LOG(2, "Read meteo grid information...");
03487
03488         /* MPTRAC meteo files... */
03489         if (ctl->clams_met_data == 0) {
03490
03491             /* Get time from filename... */
03492             size_t len = strlen(filename);
03493             sprintf(tstr, "%.4s", &filename[len - 16]);
03494             year = atoi(tstr);
03495             sprintf(tstr, "%.2s", &filename[len - 11]);
03496             mon = atoi(tstr);
03497             sprintf(tstr, "%.2s", &filename[len - 8]);

```

```

03498     day = atoi(tstr);
03499     sprintf(tstr, "%.2s", &filename[len - 5]);
03500     hour = atoi(tstr);
03501     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03502
03503     /* Check time information from data file... */
03504     if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03505         NC(nc_get_var_double(ncid, varid, &rttime));
03506         if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rttime) > 1.0)
03507             WARN("Time information in meteo file does not match filename!");
03508     } else
03509         WARN("Time information in meteo file is missing!");
03510 }
03511
03512 /* CLaMS meteo files... */
03513 else {
03514
03515     /* Read time from file... */
03516     NC_GET_DOUBLE("time", &rttime, 0);
03517
03518     /* Get time from filename (considering the century)... */
03519     if (rttime < 0)
03520         sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
03521     else
03522         sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
03523     year = atoi(tstr);
03524     sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
03525     mon = atoi(tstr);
03526     sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
03527     day = atoi(tstr);
03528     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03529     hour = atoi(tstr);
03530     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03531 }
03532
03533 /* Check time... */
03534 if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03535     || day < 1 || day > 31 || hour < 0 || hour > 23)
03536     ERRMSG("Cannot read time from filename!");
03537 jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03538 LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03539     met->time, year2, mon2, day2, hour2, min2);
03540
03541 /* Get grid dimensions... */
03542 NC_INQ_DIM("lon", &met->nx, 2, EX);
03543 LOG(2, "Number of longitudes: %d", met->nx);
03544
03545 NC_INQ_DIM("lat", &met->ny, 2, EY);
03546 LOG(2, "Number of latitudes: %d", met->ny);
03547
03548 if (ctl->vert_coord_meteo == 0) {
03549     sprintf(levname, "lev");
03550 } else {
03551     sprintf(levname, "hybrid");
03552     printf("Meteorological data is in hybrid coordinates.");
03553 }
03554 NC_INQ_DIM(levname, &met->np, 1, EP);
03555 if (met->np == 1) {
03556     int dimid;
03557     sprintf(levname, "lev_2");
03558     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03559         sprintf(levname, "plev");
03560         nc_inq_dimid(ncid, levname, &dimid);
03561     }
03562     NC(nc_inq_dimlen(ncid, dimid, &np));
03563     met->np = (int) np;
03564 }
03565 LOG(2, "Number of levels: %d", met->np);
03566 if (met->np < 2 || met->np > EP)
03567     ERRMSG("Number of levels out of range!");
03568
03569 /* Read longitudes and latitudes... */
03570 NC_GET_DOUBLE("lon", met->lon, 1);
03571 LOG(2, "Longitudes: %g, %g ... %g deg",
03572     met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03573 NC_GET_DOUBLE("lat", met->lat, 1);
03574 LOG(2, "Latitudes: %g, %g ... %g deg",
03575     met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03576
03577 /* Read pressure levels... */
03578 if (ctl->met_np <= 0) {
03579     NC_GET_DOUBLE(levname, met->p, 1);
03580     for (int ip = 0; ip < met->np; ip++)
03581         met->p[ip] /= 100.;
03582     LOG(2, "Altitude levels: %g, %g ... %g km",
03583         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03584     LOG(2, "Pressure levels: %g, %g ... %g hPa",

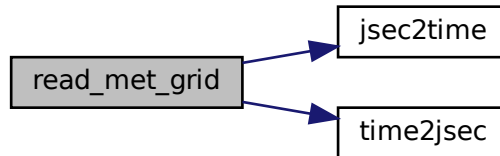
```

```

03585         met->p[0], met->p[1], met->p[met->np - 1]);
03586     }
03587 }

```

Here is the call graph for this function:



**5.19.2.46 read\_met\_levels()** void read\_met\_levels (  
     int ncid,  
     ctl\_t \* ctl,  
     met\_t \* met )

Read meteo data on vertical levels.

Definition at line 3591 of file libtrac.c.

```

03594     {
03595
03596         /* Set timer... */
03597         SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03598         LOG(2, "Read level data...");
03599
03600         /* MPTRAC meteo data... */
03601         if (ctl->clams_met_data == 0) {
03602
03603             /* Read meteo data... */
03604             if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
03605                 ERRMSG("Cannot read temperature!");
03606             if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03607                 ERRMSG("Cannot read zonal wind!");
03608             if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03609                 ERRMSG("Cannot read meridional wind!");
03610             if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
03611                 WARN("Cannot read vertical velocity!");
03612             if (!read_met_nc_3d
03613                 (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03614                 WARN("Cannot read specific humidity!");
03615             if (!read_met_nc_3d
03616                 (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03617                 WARN("Cannot read ozone data!");
03618             if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03619                 if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03620                     WARN("Cannot read cloud liquid water content!");
03621                 if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03622                     WARN("Cannot read cloud ice water content!");
03623             }
03624             if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03625                 if (!read_met_nc_3d
03626                     (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03627                     ctl->met_cloud == 2))
03628                     WARN("Cannot read cloud rain water content!");
03629                 if (!read_met_nc_3d
03630                     (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03631                     ctl->met_cloud == 2))
03632                     WARN("Cannot read cloud snow water content!");
03633             }
03634

```



```

03635      /* Transfer from model levels to pressure levels... */
03636      if (ctl->met_np > 0) {
03637
03638          /* Read pressure on model levels... */
03639          if (!read_met_nc_3d(ncid, "p1", "PL", ctl, met, met->p1, 0.01f, 1))
03640              ERRMSG("Cannot read pressure on model levels!");
03641
03642          /* Vertical interpolation from model to pressure levels... */
03643          read_met_ml2pl(ctl, met, met->t);
03644          read_met_ml2pl(ctl, met, met->u);
03645          read_met_ml2pl(ctl, met, met->v);
03646          read_met_ml2pl(ctl, met, met->w);
03647          read_met_ml2pl(ctl, met, met->h2o);
03648          read_met_ml2pl(ctl, met, met->o3);
03649          read_met_ml2pl(ctl, met, met->lwc);
03650          read_met_ml2pl(ctl, met, met->iwc);
03651
03652          /* Set new pressure levels... */
03653          met->np = ctl->met_np;
03654          for (int ip = 0; ip < met->np; ip++)
03655              met->p[ip] = ctl->met_p[ip];
03656      }
03657  }
03658
03659  /* CLaMS meteo data... */
03660  else if (ctl->clams_meteo_data == 1) {
03661
03662      /* Read meteorological data... */
03663      if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
03664          ERRMSG("Cannot read temperature!");
03665      if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03666          ERRMSG("Cannot read zonal wind!");
03667      if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03668          ERRMSG("Cannot read meridional wind!");
03669      if (!read_met_nc_3d(ncid, "w", "OMEGA", ctl, met, met->w, 0.01f, 1))
03670          WARN("Cannot read vertical velocity!");
03671      if (!read_met_nc_3d(ncid, "ZETA", "zeta", ctl, met, met->zeta, 1.0, 1))
03672          WARN("Cannot read ZETA in meteo data!");
03673      if (ctl->vert_vel == 1) {
03674          if (!read_met_nc_3d
03675              (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
03676               0.00001157407f, 1)) {
03677              if (!read_met_nc_3d
03678                  (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
03679                   0.00001157407f, 1)) {
03680                  WARN("Cannot read vertical velocity!");
03681              }
03682          }
03683      }
03684
03685      if (!read_met_nc_3d
03686          (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03687          WARN("Cannot read specific humidity!");
03688      if (!read_met_nc_3d
03689          (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03690          WARN("Cannot read ozone data!");
03691      if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03692          if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03693              WARN("Cannot read cloud liquid water content!");
03694          if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03695              WARN("Cannot read cloud ice water content!");
03696      }
03697      if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03698          if (!read_met_nc_3d
03699              (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03700               ctl->met_cloud == 2))
03701              WARN("Cannot read cloud rain water content!");
03702          if (!read_met_nc_3d
03703              (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03704               ctl->met_cloud == 2))
03705              WARN("Cannot read cloud snow water content!");
03706      }
03707
03708      /* Transfer from model levels to pressure levels... */
03709      if (ctl->met_np > 0) {
03710
03711          /* Read pressure on model levels... */
03712          if (!read_met_nc_3d(ncid, "p1", "PRESS", ctl, met, met->p1, 1.0, 1))
03713              ERRMSG("Cannot read pressure on model levels!");
03714
03715          /* Vertical interpolation from model to pressure levels... */
03716          read_met_ml2pl(ctl, met, met->t);
03717          read_met_ml2pl(ctl, met, met->u);
03718          read_met_ml2pl(ctl, met, met->v);
03719          read_met_ml2pl(ctl, met, met->w);
03720          read_met_ml2pl(ctl, met, met->h2o);
03721          read_met_ml2pl(ctl, met, met->o3);

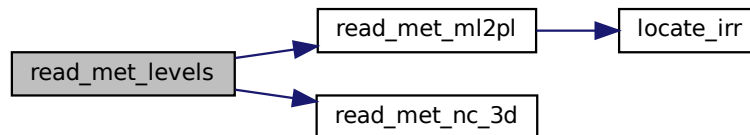
```

```

03722     read_met_ml2pl(ctl, met, met->lwc);
03723     read_met_ml2pl(ctl, met, met->iwc);
03724     if (ctl->vert_vel == 1) {
03725         read_met_ml2pl(ctl, met, met->zeta);
03726         read_met_ml2pl(ctl, met, met->zeta_dot);
03727     }
03728
03729     /* Set new pressure levels... */
03730     met->np = ctl->met_np;
03731     for (int ip = 0; ip < met->np; ip++)
03732         met->p[ip] = ctl->met_p[ip];
03733
03734     /* Create a pressure field... */
03735     for (int i = 0; i < met->nx; i++)
03736         for (int j = 0; j < met->ny; j++)
03737             for (int k = 0; k < met->np; k++) {
03738                 met->patp[i][j][k] = (float) met->p[k];
03739             }
03740 }
03741 } else
03742     ERRMSG("Meteo data format unknown!");
03743
03744 /* Check ordering of pressure levels... */
03745 for (int ip = 1; ip < met->np; ip++)
03746     if (met->p[ip - 1] < met->p[ip])
03747         ERRMSG("Pressure levels must be descending!");
03748 }

```

Here is the call graph for this function:



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

```

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

```

Convert meteo data from model levels to pressure levels.

Definition at line 3752 of file libtrac.c.

```

03755     {
03756
03757     double aux[EP], p[EP];
03758
03759     /* Set timer... */
03760     SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03761     LOG(2, "Interpolate meteo data to pressure levels...");
03762
03763     /* Loop over columns... */
03764     #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03765     for (int ix = 0; ix < met->nx; ix++)
03766         for (int iy = 0; iy < met->ny; iy++) {
03767
03768         /* Copy pressure profile... */
03769         for (int ip = 0; ip < met->np; ip++)
03770             p[ip] = met->p[ix][iy][ip];
03771
03772         /* Interpolate... */
03773         for (int ip = 0; ip < ctl->met_np; ip++) {
03774             double pt = ctl->met_p[ip];

```

```

03775         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03776             pt = p[0];
03777         else if ((pt > p[met->np - 1] && p[1] > p[0])
03778             || (pt < p[met->np - 1] && p[1] < p[0]))
03779             pt = p[met->np - 1];
03780         int ip2 = locate_irr(p, met->np, pt);
03781         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03782             p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03783     }
03784
03785     /* Copy data... */
03786     for (int ip = 0; ip < ctl->met_np; ip++)
03787         var[ix][iy][ip] = (float) aux[ip];
03788 }
03789 }

```

Here is the call graph for this function:



#### 5.19.2.48 read\_met\_nc\_2d() int read\_met\_nc\_2d (

```

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

```

Read and convert 2D variable from meteo data file.

Definition at line 3793 of file libtrac.c.

```

03801     {
03802
03803     char varsel[LEN];
03804
03805     float offset, scalfac;
03806
03807     int varid;
03808
03809     /* Check if variable exists... */
03810     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03811         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03812             WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
03813             return 0;
03814         } else {
03815             sprintf(varsel, "%s", varname2);
03816         } else
03817             sprintf(varsel, "%s", varname);
03818
03819     /* Read packed data... */
03820     if (ctl->met_nc_scale
03821         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03822         && nc_get_att_float(ncid, varid, "scale_factor",
03823             &scalfac) == NC_NOERR) {
03824
03825     /* Allocate... */
03826     short *help;
03827     ALLOC(help, short,

```

```

03828         EX * EY * EP);
03829
03830     /* Read fill value and missing value... */
03831     short fillval, missval;
03832     if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03833         fillval = 0;
03834     if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03835         missval = 0;
03836
03837     /* Write info... */
03838     LOG(2, "Read 2-D variable: %s"
03839         " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03840         varsel, fillval, missval, scalfac, offset);
03841
03842     /* Read data... */
03843     NC(nc_get_var_short(ncid, varid, help));
03844
03845     /* Copy and check data... */
03846 #pragma omp parallel for default(shared) num_threads(12)
03847     for (int ix = 0; ix < met->nx; ix++)
03848         for (int iy = 0; iy < met->ny; iy++) {
03849             if (init)
03850                 dest[ix][iy] = 0;
03851             short aux = help[ARRAY_2D(iy, ix, met->nx)];
03852             if ((fillval == 0 || aux != fillval)
03853                 && (missval == 0 || aux != missval)
03854                 && fabsf(aux * scalfac + offset) < 1e14f)
03855                 dest[ix][iy] += scl * (aux * scalfac + offset);
03856             else
03857                 dest[ix][iy] = GSL_NAN;
03858         }
03859
03860     /* Free... */
03861     free(help);
03862 }
03863
03864 /* Unpacked data... */
03865 else {
03866
03867     /* Allocate... */
03868     float *help;
03869     ALLOC(help, float,
03870         EX * EY);
03871
03872     /* Read fill value and missing value... */
03873     float fillval, missval;
03874     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03875         fillval = 0;
03876     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03877         missval = 0;
03878
03879     /* Write info... */
03880     LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03881         varsel, fillval, missval);
03882
03883     /* Read data... */
03884     NC(nc_get_var_float(ncid, varid, help));
03885
03886     /* Copy and check data... */
03887 #pragma omp parallel for default(shared) num_threads(12)
03888     for (int ix = 0; ix < met->nx; ix++)
03889         for (int iy = 0; iy < met->ny; iy++) {
03890             if (init)
03891                 dest[ix][iy] = 0;
03892             float aux = help[ARRAY_2D(iy, ix, met->nx)];
03893             if ((fillval == 0 || aux != fillval)
03894                 && (missval == 0 || aux != missval)
03895                 && fabsf(aux) < 1e14f)
03896                 dest[ix][iy] += scl * aux;
03897             else
03898                 dest[ix][iy] = GSL_NAN;
03899         }
03900
03901     /* Free... */
03902     free(help);
03903 }
03904
03905 /* Return... */
03906 return 1;
03907 }

```

**5.19.2.49 read\_met\_nc\_3d()** int read\_met\_nc\_3d (  
     int ncid,

```

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

```

Read and convert 3D variable from meteo data file.

Definition at line 3911 of file libtrac.c.

```

3919     {
3920
3921     char varsel[LEN];
3922
3923     float offset, scalfac;
3924
3925     int varid;
3926
3927     /* Check if variable exists... */
3928     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
3929         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
3930             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
3931             return 0;
3932         } else {
3933             sprintf(varsel, "%s", varname2);
3934         } else
3935             sprintf(varsel, "%s", varname);
3936
3937     /* Read packed data... */
3938     if (ctl->met_nc_scale
3939         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
3940         && nc_get_att_float(ncid, varid, "scale_factor",
3941                             &scalfac) == NC_NOERR) {
3942
3943         /* Allocate... */
3944         short *help;
3945         ALLOC(help, short,
3946              EX * EY * EP);
3947
3948         /* Read fill value and missing value... */
3949         short fillval, missval;
3950         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
3951             fillval = 0;
3952         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
3953             missval = 0;
3954
3955         /* Write info... */
3956         LOG(2, "Read 3-D variable: %s "
3957             "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
3958            varsel, fillval, missval, scalfac, offset);
3959
3960         /* Read data... */
3961         NC(nc_get_var_short(ncid, varid, help));
3962
3963         /* Copy and check data... */
3964         #pragma omp parallel for default(shared) num_threads(12)
3965         for (int ix = 0; ix < met->nx; ix++)
3966             for (int iy = 0; iy < met->ny; iy++)
3967                 for (int ip = 0; ip < met->np; ip++) {
3968                     if (init)
3969                         dest[ix][iy][ip] = 0;
3970                     short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
3971                     if ((fillval == 0 || aux != fillval)
3972                         && (missval == 0 || aux != missval)
3973                         && fabsf(aux * scalfac + offset) < 1e14f)
3974                         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
3975                     else
3976                         dest[ix][iy][ip] = GSL_NAN;
3977                 }
3978
3979         /* Free... */
3980         free(help);
3981     }
3982
3983     /* Unpacked data... */
3984     else {
3985
3986         /* Allocate... */
3987         float *help;
3988         ALLOC(help, float,
3989              EX * EY * EP);

```

```

03990
03991      /* Read fill value and missing value... */
03992      float fillval, missval;
03993      if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03994          fillval = 0;
03995      if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03996          missval = 0;
03997
03998      /* Write info... */
03999      LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
04000          varsel, fillval, missval);
04001
04002      /* Read data... */
04003      NC(nc_get_var_float(ncid, varid, help));
04004
04005      /* Copy and check data... */
04006      #pragma omp parallel for default(shared) num_threads(12)
04007      for (int ix = 0; ix < met->nx; ix++)
04008          for (int iy = 0; iy < met->ny; iy++)
04009              for (int ip = 0; ip < met->np; ip++) {
04010                  if (init)
04011                      dest[ix][iy][ip] = 0;
04012                  float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04013                  if ((fillval == 0 || aux != fillval)
04014                      && (missval == 0 || aux != missval)
04015                      && fabsf(aux) < 1e14f)
04016                      dest[ix][iy][ip] += scl * aux;
04017                  else
04018                      dest[ix][iy][ip] = GSL_NAN;
04019              }
04020
04021      /* Free... */
04022      free(help);
04023  }
04024
04025      /* Return... */
04026      return 1;
04027  }

```

**5.19.2.50 read\_met\_pbl()** void read\_met\_pbl (  
     met\_t \* met )

Calculate pressure of the boundary layer.

Definition at line 4031 of file libtrac.c.

```

04032      {
04033
04034      /* Set timer... */
04035      SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
04036      LOG(2, "Calculate planetary boundary layer...");
04037
04038      /* Parameters used to estimate the height of the PBL
04039       (e.g., Voegelezang and Holtslag, 1996; Seidel et al., 2012)... */
04040      const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
04041
04042      /* Loop over grid points... */
04043      #pragma omp parallel for default(shared) collapse(2)
04044      for (int ix = 0; ix < met->nx; ix++)
04045          for (int iy = 0; iy < met->ny; iy++) {
04046
04047              /* Set bottom level of PBL... */
04048              double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
04049
04050              /* Find lowest level near the bottom... */
04051              int ip;
04052              for (ip = 1; ip < met->np; ip++)
04053                  if (met->p[ip] < pbl_bot)
04054                      break;
04055
04056              /* Get near surface data... */
04057              double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
04058                             met->p[ip], met->z[ix][iy][ip], pbl_bot);
04059              double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
04060                             met->p[ip], met->t[ix][iy][ip], pbl_bot);
04061              double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
04062                             met->p[ip], met->u[ix][iy][ip], pbl_bot);
04063              double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
04064                             met->p[ip], met->v[ix][iy][ip], pbl_bot);
04065              double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],

```

```

04066         met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
04067     double tvs = THETA_VIRT(pbl_bot, ts, h2os);
04068
04069     /* Init... */
04070     double rib_old = 0;
04071
04072     /* Loop over levels... */
04073     for (; ip < met->np; ip++) {
04074
04075         /* Get squared horizontal wind speed... */
04076         double vh2
04077             = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
04078         vh2 = GSL_MAX(vh2, SQR(umin));
04079
04080         /* Calculate bulk Richardson number... */
04081         double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
04082             * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
04083                 met->h2o[ix][iy][ip]) - tvs) / vh2;
04084
04085         /* Check for critical value... */
04086         if (rib >= rib_crit) {
04087             met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
04088                 rib, met->p[ip], rib_crit));
04089             if (met->pbl[ix][iy] > pbl_bot)
04090                 met->pbl[ix][iy] = (float) pbl_bot;
04091             break;
04092         }
04093
04094         /* Save Richardson number... */
04095         rib_old = rib;
04096     }
04097 }
04098 }

```

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

Create meteo data with periodic boundary conditions.

Definition at line 4102 of file libtrac.c.

```

04103     {
04104
04105         /* Set timer... */
04106         SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
04107         LOG(2, "Apply periodic boundary conditions...");
04108
04109         /* Check longitudes... */
04110         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
04111             + met->lon[1] - met->lon[0] - 360) < 0.01))
04112             return;
04113
04114         /* Increase longitude counter... */
04115         if ((++met->nx) > EX)
04116             ERRMSG("Cannot create periodic boundary conditions!");
04117
04118         /* Set longitude... */
04119         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
04120
04121         /* Loop over latitudes and pressure levels... */
04122         #pragma omp parallel for default(shared)
04123         for (int iy = 0; iy < met->ny; iy++) {
04124             met->ps[met->nx - 1][iy] = met->ps[0][iy];
04125             met->zs[met->nx - 1][iy] = met->zs[0][iy];
04126             met->ts[met->nx - 1][iy] = met->ts[0][iy];
04127             met->us[met->nx - 1][iy] = met->us[0][iy];
04128             met->vs[met->nx - 1][iy] = met->vs[0][iy];
04129             for (int ip = 0; ip < met->np; ip++) {
04130                 met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
04131                 met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
04132                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
04133                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
04134                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
04135                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
04136                 met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
04137                 met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
04138             }
04139         }
04140     }

```

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

Calculate potential vorticity.

Definition at line 4144 of file libtrac.c.

```

04145     {
04146
04147     double pows[EP];
04148
04149     /* Set timer... */
04150     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
04151     LOG(2, "Calculate potential vorticity...");
04152
04153     /* Set powers... */
04154     #pragma omp parallel for default(shared)
04155     for (int ip = 0; ip < met->np; ip++)
04156         pows[ip] = pow(1000. / met->p[ip], 0.286);
04157
04158     /* Loop over grid points... */
04159     #pragma omp parallel for default(shared)
04160     for (int ix = 0; ix < met->nx; ix++) {
04161
04162         /* Set indices... */
04163         int ix0 = GSL_MAX(ix - 1, 0);
04164         int ix1 = GSL_MIN(ix + 1, met->nx - 1);
04165
04166         /* Loop over grid points... */
04167         for (int iy = 0; iy < met->ny; iy++) {
04168
04169             /* Set indices... */
04170             int iy0 = GSL_MAX(iy - 1, 0);
04171             int iy1 = GSL_MIN(iy + 1, met->ny - 1);
04172
04173             /* Set auxiliary variables... */
04174             double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
04175             double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
04176             double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
04177             double c0 = cos(met->lat[iy0] / 180. * M_PI);
04178             double c1 = cos(met->lat[iy1] / 180. * M_PI);
04179             double cr = cos(latr / 180. * M_PI);
04180             double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
04181
04182             /* Loop over grid points... */
04183             for (int ip = 0; ip < met->np; ip++) {
04184
04185                 /* Get gradients in longitude... */
04186                 double dtdx
04187                     = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
04188                 double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
04189
04190                 /* Get gradients in latitude... */
04191                 double dtdy
04192                     = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
04193                 double dudx
04194                     = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
04195
04196                 /* Set indices... */
04197                 int ip0 = GSL_MAX(ip - 1, 0);
04198                 int ip1 = GSL_MIN(ip + 1, met->np - 1);
04199
04200                 /* Get gradients in pressure... */
04201                 double dtdp, dudp, dvdp;
04202                 double dp0 = 100. * (met->p[ip] - met->p[ip0]);
04203                 double dp1 = 100. * (met->p[ip1] - met->p[ip]);
04204                 if (ip != ip0 && ip != ip1) {
04205                     double denom = dp0 * dp1 * (dp0 + dp1);
04206                     dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
04207                         - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
04208                         + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
04209                         / denom;
04210                     dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
04211                         - dp1 * dp1 * met->u[ix][iy][ip0]
04212                         + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
04213                         / denom;
04214                     dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
04215                         - dp1 * dp1 * met->v[ix][iy][ip0]
04216                         + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
04217                         / denom;
04218                 } else {
04219                     double denom = dp0 + dp1;
04220                     dtdp =
04221                         (met->t[ix][iy][ip1] * pows[ip1] -
04222                         met->t[ix][iy][ip0] * pows[ip0]) / denom;
04223                     dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;

```



```

04224         dvdp = (met->v[ix][iy][ipl] - met->v[ix][iy][ip0]) / denom;
04225     }
04226
04227     /* Calculate PV... */
04228     met->pv[ix][iy][ip] = (float)
04229         (le6 * G0 *
04230          (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
04231     }
04232 }
04233 }
04234
04235 /* Fix for polar regions... */
04236 #pragma omp parallel for default(shared)
04237 for (int ix = 0; ix < met->nx; ix++)
04238     for (int ip = 0; ip < met->np; ip++) {
04239         met->pv[ix][0][ip]
04240             = met->pv[ix][1][ip]
04241             = met->pv[ix][2][ip];
04242         met->pv[ix][met->ny - 1][ip]
04243             = met->pv[ix][met->ny - 2][ip]
04244             = met->pv[ix][met->ny - 3][ip];
04245     }
04246 }

```

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

Downsampling of meteo data.

Definition at line 4250 of file libtrac.c.

```

04252     {
04253
04254         met_t *help;
04255
04256         /* Check parameters... */
04257         if (ctl->met_dx <= 1 && ctl->met_dy <= 1 && ctl->met_dx <= 1 &&
04258             && ctl->met_sy <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
04259             return;
04260
04261         /* Set timer... */
04262         SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
04263         LOG(2, "Downsampling of meteo data...");
04264
04265         /* Allocate... */
04266         ALLOC(help, met_t, 1);
04267
04268         /* Copy data... */
04269         help->nx = met->nx;
04270         help->ny = met->ny;
04271         help->np = met->np;
04272         memcpy(help->lon, met->lon, sizeof(met->lon));
04273         memcpy(help->lat, met->lat, sizeof(met->lat));
04274         memcpy(help->p, met->p, sizeof(met->p));
04275
04276         /* Smoothing... */
04277         for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
04278             for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
04279                 for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
04280                     help->ps[ix][iy] = 0;
04281                     help->zs[ix][iy] = 0;
04282                     help->ts[ix][iy] = 0;
04283                     help->us[ix][iy] = 0;
04284                     help->vs[ix][iy] = 0;
04285                     help->t[ix][iy][ip] = 0;
04286                     help->u[ix][iy][ip] = 0;
04287                     help->v[ix][iy][ip] = 0;
04288                     help->w[ix][iy][ip] = 0;
04289                     help->h2o[ix][iy][ip] = 0;
04290                     help->o3[ix][iy][ip] = 0;
04291                     help->lwc[ix][iy][ip] = 0;
04292                     help->iwc[ix][iy][ip] = 0;
04293                     float wsum = 0;
04294                     for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
04295                         ix2++) {
04296                         int ix3 = ix2;
04297                         if (ix3 < 0)
04298                             ix3 += met->nx;

```

```

04299     else if (ix3 >= met->nx)
04300         ix3 -= met->nx;
04301
04302     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
04303          iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
04304         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
04305              ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
04306             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
04307                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
04308                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
04309             help->ps[ix][iy] += w * met->ps[ix3][iy2];
04310             help->zs[ix][iy] += w * met->zs[ix3][iy2];
04311             help->ts[ix][iy] += w * met->ts[ix3][iy2];
04312             help->us[ix][iy] += w * met->us[ix3][iy2];
04313             help->vs[ix][iy] += w * met->vs[ix3][iy2];
04314             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
04315             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
04316             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
04317             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
04318             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
04319             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
04320             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
04321             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
04322             wsum += w;
04323         }
04324     }
04325     help->ps[ix][iy] /= wsum;
04326     help->zs[ix][iy] /= wsum;
04327     help->ts[ix][iy] /= wsum;
04328     help->us[ix][iy] /= wsum;
04329     help->vs[ix][iy] /= wsum;
04330     help->t[ix][iy][ip] /= wsum;
04331     help->u[ix][iy][ip] /= wsum;
04332     help->v[ix][iy][ip] /= wsum;
04333     help->w[ix][iy][ip] /= wsum;
04334     help->h2o[ix][iy][ip] /= wsum;
04335     help->o3[ix][iy][ip] /= wsum;
04336     help->lwc[ix][iy][ip] /= wsum;
04337     help->iwc[ix][iy][ip] /= wsum;
04338 }
04339 }
04340 }
04341
04342 /* Downsampling... */
04343 met->nx = 0;
04344 for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04345     met->lon[met->nx] = help->lon[ix];
04346     met->ny = 0;
04347     for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04348         met->lat[met->ny] = help->lat[iy];
04349         met->ps[met->nx][met->ny] = help->ps[ix][iy];
04350         met->zs[met->nx][met->ny] = help->zs[ix][iy];
04351         met->ts[met->nx][met->ny] = help->ts[ix][iy];
04352         met->us[met->nx][met->ny] = help->us[ix][iy];
04353         met->vs[met->nx][met->ny] = help->vs[ix][iy];
04354         met->np = 0;
04355         for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04356             met->p[met->np] = help->p[ip];
04357             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04358             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04359             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04360             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04361             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04362             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04363             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
04364             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04365             met->np++;
04366         }
04367         met->ny++;
04368     }
04369     met->nx++;
04370 }
04371
04372 /* Free... */
04373 free(help);
04374 }

```

```

5.19.2.54 read_met_surface() void read_met_surface (
    int ncid,
    met_t * met,
    ctl_t * ctl )

```

Read surface data.

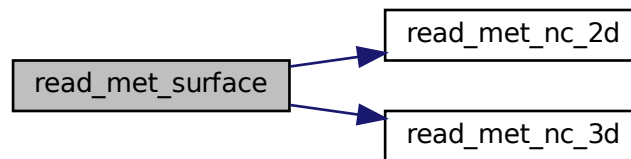
Definition at line 4378 of file libtrac.c.

```

04381     {
04382
04383     /* Set timer... */
04384     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04385     LOG(2, "Read surface data...");
04386
04387     /* MPTRAC meteo data... */
04388     if (ctl->clams_met_data == 0) {
04389
04390         /* Read surface pressure... */
04391         if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
04392             if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04393                 WARN("Cannot not read surface pressure data (use lowest level!)");
04394                 for (int ix = 0; ix < met->nx; ix++)
04395                     for (int iy = 0; iy < met->ny; iy++)
04396                         met->ps[ix][iy] = (float) met->p[0];
04397             }
04398         } else
04399             for (int ix = 0; ix < met->nx; ix++)
04400                 for (int iy = 0; iy < met->ny; iy++)
04401                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04402
04403         /* Read geopotential height at the surface... */
04404         if (!read_met_nc_2d(ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
04405             if (!read_met_nc_2d(ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
04406                 WARN("Cannot read surface geopotential height!");
04407
04408         /* Read temperature at the surface... */
04409         if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
04410             WARN("Cannot read surface temperature!");
04411
04412         /* Read zonal wind at the surface... */
04413         if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
04414             WARN("Cannot read surface zonal wind!");
04415
04416         /* Read meridional wind at the surface... */
04417         if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
04418             WARN("Cannot read surface meridional wind!");
04419     }
04420
04421     /* CLaMS meteo data... */
04422     else {
04423
04424         /* Read surface pressure... */
04425         if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04426             WARN("Cannot not read surface pressure data (use lowest level!)");
04427             for (int ix = 0; ix < met->nx; ix++)
04428                 for (int iy = 0; iy < met->ny; iy++)
04429                     met->ps[ix][iy] = (float) met->p[0];
04430         }
04431
04432         /* Read geopotential height at the surface
04433            (use lowermost level of 3-D data field)... */
04434         float *help;
04435         ALLOC(help, float,
04436              EX * EY * EP);
04437         memcpy(help, met->p1, sizeof(met->p1));
04438         if (!read_met_nc_3d(ncid, "gph", "GPH", ctl, met, met->p1, (float) (1e-3 / G0), 1)) {
04439             ERRMSG("Cannot read geopotential height!");
04440         } else
04441             for (int ix = 0; ix < met->nx; ix++)
04442                 for (int iy = 0; iy < met->ny; iy++)
04443                     met->zs[ix][iy] = met->p1[ix][iy][0];
04444         memcpy(met->p1, help, sizeof(met->p1));
04445         free(help);
04446
04447         /* Read temperature at the surface... */
04448         if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
04449             WARN("Cannot read surface temperature!");
04450
04451         /* Read zonal wind at the surface... */
04452         if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
04453             WARN("Cannot read surface zonal wind!");
04454
04455         /* Read meridional wind at the surface... */
04456         if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
04457             WARN("Cannot read surface meridional wind!");
04458     }
04459 }
04460
04461 }
04462 }

```

Here is the call graph for this function:



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

Calculate tropopause data.

Definition at line 4466 of file libtrac.c.

```

04469     {
04470
04471     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04472           th2[200], z[EP], z2[200];
04473
04474     /* Set timer... */
04475     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04476     LOG(2, "Calculate tropopause...");
04477
04478     /* Get altitude and pressure profiles... */
04479     #pragma omp parallel for default(shared)
04480     for (int iz = 0; iz < met->np; iz++)
04481         z[iz] = Z(met->p[iz]);
04482     #pragma omp parallel for default(shared)
04483     for (int iz = 0; iz <= 190; iz++) {
04484         z2[iz] = 4.5 + 0.1 * iz;
04485         p2[iz] = P(z2[iz]);
04486     }
04487
04488     /* Do not calculate tropopause... */
04489     if (ctl->met_tropo == 0)
04490     #pragma omp parallel for default(shared) collapse(2)
04491     for (int ix = 0; ix < met->nx; ix++)
04492         for (int iy = 0; iy < met->ny; iy++)
04493             met->pt[ix][iy] = GSL_NAN;
04494
04495     /* Use tropopause climatology... */
04496     else if (ctl->met_tropo == 1) {
04497     #pragma omp parallel for default(shared) collapse(2)
04498     for (int ix = 0; ix < met->nx; ix++)
04499         for (int iy = 0; iy < met->ny; iy++)
04500             met->pt[ix][iy] = (float) clim_tropo(clim, met->time, met->lat[iy]);
04501     }
04502
04503     /* Use cold point... */
04504     else if (ctl->met_tropo == 2) {
04505
04506         /* Loop over grid points... */
04507     #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04508     for (int ix = 0; ix < met->nx; ix++)
04509         for (int iy = 0; iy < met->ny; iy++) {
04510
04511             /* Interpolate temperature profile... */
04512             for (int iz = 0; iz < met->np; iz++)
04513                 t[iz] = met->t[ix][iy][iz];
04514             spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);

```

```

04515
04516     /* Find minimum... */
04517     int iz = (int) gsl_stats_min_index(t2, 1, 171);
04518     if (iz > 0 && iz < 170)
04519         met->pt[ix][iy] = (float) p2[iz];
04520     else
04521         met->pt[ix][iy] = GSL_NAN;
04522 }
04523 }
04524
04525 /* Use WMO definition... */
04526 else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04527
04528     /* Loop over grid points... */
04529     #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04530     for (int ix = 0; ix < met->nx; ix++)
04531         for (int iy = 0; iy < met->ny; iy++) {
04532
04533             /* Interpolate temperature profile... */
04534             int iz;
04535             for (iz = 0; iz < met->np; iz++)
04536                 t[iz] = met->t[ix][iy][iz];
04537             spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04538
04539             /* Find 1st tropopause... */
04540             met->pt[ix][iy] = GSL_NAN;
04541             for (iz = 0; iz <= 170; iz++) {
04542                 int found = 1;
04543                 for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04544                     if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04545                         ctl->met_tropo_lapse) {
04546                         found = 0;
04547                         break;
04548                     }
04549                 if (found) {
04550                     if (iz > 0 && iz < 170)
04551                         met->pt[ix][iy] = (float) p2[iz];
04552                     break;
04553                 }
04554             }
04555
04556             /* Find 2nd tropopause... */
04557             if (ctl->met_tropo == 4) {
04558                 met->pt[ix][iy] = GSL_NAN;
04559                 for (; iz <= 170; iz++) {
04560                     int found = 1;
04561                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04562                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04563                             ctl->met_tropo_lapse_sep) {
04564                             found = 0;
04565                             break;
04566                         }
04567                     if (found)
04568                         break;
04569                 }
04570                 for (; iz <= 170; iz++) {
04571                     int found = 1;
04572                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04573                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04574                             ctl->met_tropo_lapse) {
04575                             found = 0;
04576                             break;
04577                         }
04578                     if (found) {
04579                         if (iz > 0 && iz < 170)
04580                             met->pt[ix][iy] = (float) p2[iz];
04581                         break;
04582                     }
04583                 }
04584             }
04585         }
04586     }
04587
04588     /* Use dynamical tropopause... */
04589     else if (ctl->met_tropo == 5) {
04590
04591         /* Loop over grid points... */
04592         #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04593         for (int ix = 0; ix < met->nx; ix++)
04594             for (int iy = 0; iy < met->ny; iy++) {
04595
04596                 /* Interpolate potential vorticity profile... */
04597                 for (int iz = 0; iz < met->np; iz++)
04598                     pv[iz] = met->pv[ix][iy][iz];
04599                 spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04600
04601                 /* Interpolate potential temperature profile... */

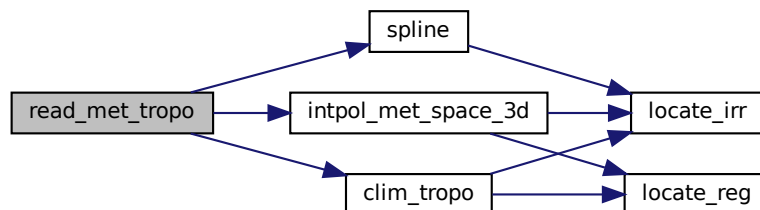
```

```

04602     for (int iz = 0; iz < met->np; iz++)
04603     {
04604         th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04605         spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04606
04607         /* Find dynamical tropopause... */
04608         met->pt[ix][iy] = GSL_NAN;
04609         for (int iz = 0; iz <= 170; iz++)
04610             if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04611                 || th2[iz] >= ctl->met_tropo_theta) {
04612                 if (iz > 0 && iz < 170)
04613                     met->pt[ix][iy] = (float) p2[iz];
04614                 break;
04615             }
04616     }
04617
04618     else
04619         ERRMSG("Cannot calculate tropopause!");
04620
04621     /* Interpolate temperature, geopotential height, and water vapor vmr... */
04622     #pragma omp parallel for default(shared) collapse(2)
04623     for (int ix = 0; ix < met->nx; ix++)
04624         for (int iy = 0; iy < met->ny; iy++) {
04625             double h2ot, tt, zt;
04626             INTPOL_INIT;
04627             intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04628                               met->lat[iy], &tt, ci, cw, 1);
04629             intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04630                               met->lat[iy], &zt, ci, cw, 0);
04631             intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04632                               met->lat[iy], &h2ot, ci, cw, 0);
04633             met->tt[ix][iy] = (float) tt;
04634             met->zt[ix][iy] = (float) zt;
04635             met->h2ot[ix][iy] = (float) h2ot;
04636         }
04637 }

```

Here is the call graph for this function:



**5.19.2.56 read\_obs()** void read\_obs (

```

    char * filename,
    double * rt,
    double * rz,
    double * rlon,
    double * rlat,
    double * robs,
    int * nobss )

```

Read observation data.

Definition at line 4641 of file libtrac.c.

```

04648 {
04649

```

```

04650 FILE *in;
04651
04652 char line[LEN];
04653
04654 /* Open observation data file... */
04655 LOG(1, "Read observation data: %s", filename);
04656 if (!(in = fopen(filename, "r")))
04657     ERRMSG("Cannot open file!");
04658
04659 /* Read observations... */
04660 while (fgets(line, LEN, in))
04661     if (sscanf(line, "%lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
04662               &rln[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
04663         if ((++(*nobs)) >= NOBS)
04664             ERRMSG("Too many observations!");
04665
04666 /* Close observation data file... */
04667 fclose(in);
04668
04669 /* Check time... */
04670 for (int i = 1; i < *nobs; i++)
04671     if (rt[i] < rt[i - 1])
04672         ERRMSG("Time must be ascending!");
04673
04674 /* Write info... */
04675 int n = *nobs;
04676 double mini, maxi;
04677 LOG(2, "Number of observations: %d", *nobs);
04678 gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
04679 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04680 gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
04681 LOG(2, "Altitude range: %g ... %g km", mini, maxi);
04682 gsl_stats_minmax(&mini, &maxi, rln, 1, (size_t) n);
04683 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04684 gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
04685 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04686 gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
04687 LOG(2, "Observation range: %g ... %g", mini, maxi);
04688 }

```

**5.19.2.57 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 4692 of file libtrac.c.

```

04699 {
04700
04701 FILE *in = NULL;
04702
04703 char fullname1[LEN], fullname2[LEN], rval[LEN];
04704
04705 int contain = 0, i;
04706
04707 /* Open file... */
04708 if (filename[strlen(filename) - 1] != '-')
04709     if (!(in = fopen(filename, "r")))
04710         ERRMSG("Cannot open file!");
04711
04712 /* Set full variable name... */
04713 if (arridx >= 0) {
04714     sprintf(fullname1, "%s[%d]", varname, arridx);
04715     sprintf(fullname2, "%s[*]", varname);
04716 } else {
04717     sprintf(fullname1, "%s", varname);
04718     sprintf(fullname2, "%s", varname);
04719 }
04720
04721 /* Read data... */
04722 if (in != NULL) {

```

```

04723     char dummy[LEN], line[LEN], rvarname[LEN];
04724     while (fgets(line, LEN, in)) {
04725         if (sscanf(line, "%4999s %4999s", rvarname, dummy, rval) == 3)
04726             if (strcascmp(rvarname, fullname1) == 0 ||
04727                 strcascmp(rvarname, fullname2) == 0) {
04728                 contain = 1;
04729                 break;
04730             }
04731     }
04732 }
04733 for (i = 1; i < argc - 1; i++)
04734     if (strcascmp(argv[i], fullname1) == 0 ||
04735         strcascmp(argv[i], fullname2) == 0) {
04736         sprintf(rval, "%s", argv[i + 1]);
04737         contain = 1;
04738         break;
04739     }
04740
04741 /* Close file... */
04742 if (in != NULL)
04743     fclose(in);
04744
04745 /* Check for missing variables... */
04746 if (!contain) {
04747     if (strlen(defvalue) > 0)
04748         sprintf(rval, "%s", defvalue);
04749     else
04750         ERRMSG("Missing variable %s!\n", fullname1);
04751 }
04752
04753 /* Write info... */
04754 LOG(1, "%s = %s", fullname1, rval);
04755
04756 /* Return values... */
04757 if (value != NULL)
04758     sprintf(value, "%s", rval);
04759 return atof(rval);
04760 }

```

**5.19.2.58 sedi()** double sedi (  
     double p,  
     double T,  
     double rp,  
     double rhop )

Calculate sedimentation velocity.

Definition at line 4764 of file libtrac.c.

```

04768     {
04769
04770     /* Convert particle radius from microns to m... */
04771     rp *= 1e-6;
04772
04773     /* Density of dry air [kg / m^3]... */
04774     double rho = RHO(p, T);
04775
04776     /* Dynamic viscosity of air [kg / (m s)]... */
04777     double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04778
04779     /* Thermal velocity of an air molecule [m / s]... */
04780     double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04781
04782     /* Mean free path of an air molecule [m]... */
04783     double lambda = 2. * eta / (rho * v);
04784
04785     /* Knudsen number for air (dimensionless)... */
04786     double K = lambda / rp;
04787
04788     /* Cunningham slip-flow correction (dimensionless)... */
04789     double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04790
04791     /* Sedimentation velocity [m / s]... */
04792     return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
04793 }

```



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

Spline interpolation.

Definition at line 4797 of file libtrac.c.

```

04804     {
04805
04806     /* Cubic spline interpolation... */
04807     if (method == 1) {
04808
04809         /* Allocate... */
04810         gsl_interp_accel *acc;
04811         gsl_spline *s;
04812         acc = gsl_interp_accel_alloc();
04813         s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04814
04815         /* Interpolate profile... */
04816         gsl_spline_init(s, x, y, (size_t) n);
04817         for (int i = 0; i < n2; i++)
04818             if (x2[i] <= x[0])
04819                 y2[i] = y[0];
04820             else if (x2[i] >= x[n - 1])
04821                 y2[i] = y[n - 1];
04822             else
04823                 y2[i] = gsl_spline_eval(s, x2[i], acc);
04824
04825         /* Free... */
04826         gsl_spline_free(s);
04827         gsl_interp_accel_free(acc);
04828     }
04829
04830     /* Linear interpolation... */
04831     else {
04832         for (int i = 0; i < n2; i++)
04833             if (x2[i] <= x[0])
04834                 y2[i] = y[0];
04835             else if (x2[i] >= x[n - 1])
04836                 y2[i] = y[n - 1];
04837             else {
04838                 int idx = locate_irr(x, n, x2[i]);
04839                 y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04840             }
04841     }
04842 }
```

Here is the call graph for this function:



**5.19.2.60 stddev()** float stddev (  
float \* data,  
int n )

Calculate standard deviation.

Definition at line 4846 of file libtrac.c.

```

04848     {
04849
04850     if (n <= 0)
04851         return 0;
04852
04853     float mean = 0, var = 0;
04854
04855     for (int i = 0; i < n; ++i) {
04856         mean += data[i];
04857         var += SQR(data[i]);
04858     }
04859
04860     var = var / (float) n - SQR(mean / (float) n);
04861
04862     return (var > 0 ? sqrtf(var) : 0);
04863 }
```

**5.19.2.61** **sza()** double sza (  
     double sec,  
     double lon,  
     double lat )

Calculate solar zenith angle.

Definition at line 4867 of file libtrac.c.

```

04870     {
04871
04872     double D, dec, e, g, GMST, h, L, LST, q, ra;
04873
04874     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
04875     D = sec / 86400 - 0.5;
04876
04877     /* Geocentric apparent ecliptic longitude [rad]... */
04878     g = (357.529 + 0.98560028 * D) * M_PI / 180;
04879     q = 280.459 + 0.98564736 * D;
04880     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
04881
04882     /* Mean obliquity of the ecliptic [rad]... */
04883     e = (23.439 - 0.00000036 * D) * M_PI / 180;
04884
04885     /* Declination [rad]... */
04886     dec = asin(sin(e) * sin(L));
04887
04888     /* Right ascension [rad]... */
04889     ra = atan2(cos(e) * sin(L), cos(L));
04890
04891     /* Greenwich Mean Sidereal Time [h]... */
04892     GMST = 18.697374558 + 24.06570982441908 * D;
04893
04894     /* Local Sidereal Time [h]... */
04895     LST = GMST + lon / 15;
04896
04897     /* Hour angle [rad]... */
04898     h = LST / 12 * M_PI - ra;
04899
04900     /* Convert latitude... */
04901     lat *= M_PI / 180;
04902
04903     /* Return solar zenith angle [rad]... */
04904     return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
04905 }
```

**5.19.2.62 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 4909 of file libtrac.c.

```

04917     {
04918
04919     struct tm t0, t1;
04920
04921     t0.tm_year = 100;
04922     t0.tm_mon = 0;
04923     t0.tm_mday = 1;
04924     t0.tm_hour = 0;
04925     t0.tm_min = 0;
04926     t0.tm_sec = 0;
04927
04928     t1.tm_year = year - 1900;
04929     t1.tm_mon = mon - 1;
04930     t1.tm_mday = day;
04931     t1.tm_hour = hour;
04932     t1.tm_min = min;
04933     t1.tm_sec = sec;
04934
04935     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04936 }
```

**5.19.2.63 timer()** void timer (

```

    const char * name,
    const char * group,
    int output )
```

Measure wall-clock time.

Definition at line 4940 of file libtrac.c.

```

04943     {
04944
04945     static char names[NTIMER][100], groups[NTIMER][100];
04946
04947     static double rt_name[NTIMER], rt_group[NTIMER],
04948         rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;
04949
04950     static int iname = -1, igrup = -1, nname, ngroup, ct_name[NTIMER];
04951
04952     /* Get time... */
04953     t1 = omp_get_wtime();
04954     dt = t1 - t0;
04955
04956     /* Add elapsed time to current timers... */
04957     if (iname >= 0) {
04958         rt_name[iname] += dt;
04959         rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
04960         rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
04961         ct_name[iname]++;
04962     }
04963     if (igrup >= 0)
04964         rt_group[igrup] += t1 - t0;
04965
04966     /* Report timers... */
04967     if (output) {
04968         for (int i = 0; i < nname; i++)
04969             LOG(1, "TIMER_%s = %.3f s (min= %g s, mean= %g s, "
04970                 " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
04971                 rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
04972         for (int i = 0; i < ngroup; i++)
```

```

04973     LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
04974     double total = 0.0;
04975     for (int i = 0; i < nname; i++)
04976         total += rt_name[i];
04977     LOG(1, "TIMER_TOTAL = %.3f s", total);
04978 }
04979
04980 /* Identify IDs of next timer... */
04981 for (iname = 0; iname < nname; iname++)
04982     if (strcasecmp(name, names[iname]) == 0)
04983         break;
04984 for (igroup = 0; igroup < ngroup; igroup++)
04985     if (strcasecmp(group, groups[igroup]) == 0)
04986         break;
04987
04988 /* Check whether this is a new timer... */
04989 if (iname >= nname) {
04990     sprintf(names[iname], "%s", name);
04991     if ((++nname) > NTIMER)
04992         ERRMSG("Too many timers!");
04993 }
04994
04995 /* Check whether this is a new group... */
04996 if (igroup >= ngroup) {
04997     sprintf(groups[igroup], "%s", group);
04998     if ((++ngroup) > NTIMER)
04999         ERRMSG("Too many groups!");
05000 }
05001
05002 /* Save starting time... */
05003 t0 = t1;
05004 }

```

**5.19.2.64 tropo\_weight()** double tropo\_weight (  
     clim\_t \* clim,  
     double t,  
     double lat,  
     double p )

Get weighting factor based on tropopause distance.

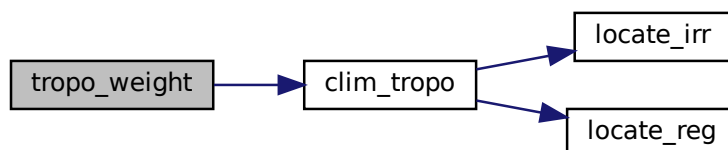
Definition at line 5008 of file libtrac.c.

```

05012     {
05013
05014     /* Get tropopause pressure... */
05015     double pt = clim_tropo(clim, t, lat);
05016
05017     /* Get pressure range... */
05018     double p1 = pt * 0.866877899;
05019     double p0 = pt / 0.866877899;
05020
05021     /* Get weighting factor... */
05022     if (p > p0)
05023         return 1;
05024     else if (p < p1)
05025         return 0;
05026     else
05027         return LIN(p0, 1.0, p1, 0.0, p);
05028 }

```

Here is the call graph for this function:



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

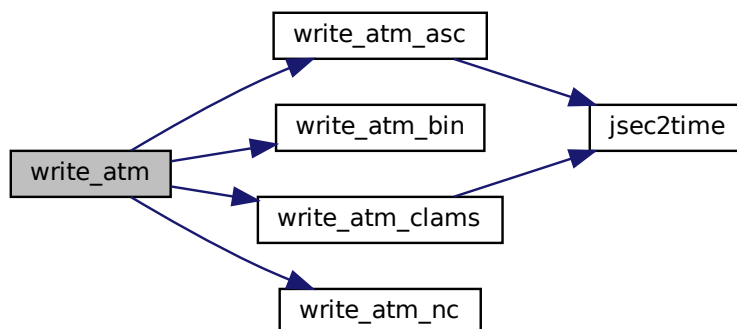
Write atmospheric data.

Definition at line 5032 of file libtrac.c.

```

05036     {
05037
05038     /* Set timer... */
05039     SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
05040
05041     /* Write info... */
05042     LOG(1, "Write atmospheric data: %s", filename);
05043
05044     /* Write ASCII data... */
05045     if (ctl->atm_type == 0)
05046         write_atm_asc(filename, ctl, atm, t);
05047
05048     /* Write binary data... */
05049     else if (ctl->atm_type == 1)
05050         write_atm_bin(filename, ctl, atm);
05051
05052     /* Write netCDF data... */
05053     else if (ctl->atm_type == 2)
05054         write_atm_nc(filename, ctl, atm);
05055
05056     /* Write CLaMS data... */
05057     else if (ctl->atm_type == 3)
05058         write_atm_clams(ctl, atm, t);
05059
05060     /* Error... */
05061     else
05062         ERRMSG("Atmospheric data type not supported!");
05063
05064     /* Write info... */
05065     double mini, maxi;
05066     LOG(2, "Number of particles: %d", atm->np);
05067     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
05068     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
05069     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
05070     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
05071     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
05072     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
05073     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
05074     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
05075     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
05076     for (int iq = 0; iq < ctl->nq; iq++) {
05077         char msg[LEN];
05078         sprintf(msg, "Quantity %s range: %s ... %s %s",
05079                 ctl->qnt_name[iq], ctl->qnt_format[iq],
05080                 ctl->qnt_format[iq], ctl->qnt_unit[iq]);
05081         gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
05082         LOG(2, msg, mini, maxi);
05083     }
05084 }
```

Here is the call graph for this function:



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

Write atmospheric data in ASCII format.

Definition at line 5088 of file libtrac.c.

```

05092     {
05093
05094     FILE *out;
05095
05096     /* Set time interval for output... */
05097     double t0 = t - 0.5 * ctl->dt_mod;
05098     double t1 = t + 0.5 * ctl->dt_mod;
05099
05100     /* Check if gnuplot output is requested... */
05101     if (ctl->atm_gpfile[0] != '-') {
05102
05103         /* Create gnuplot pipe... */
05104         if (!(out = popen("gnuplot", "w")))
05105             ERRMSG("Cannot create pipe to gnuplot!");
05106
05107         /* Set plot filename... */
05108         fprintf(out, "set out \"%s.png\"\n", filename);
05109
05110         /* Set time string... */
05111         double r;
05112         int year, mon, day, hour, min, sec;
05113         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05114         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05115             year, mon, day, hour, min);
05116
05117         /* Dump gnuplot file to pipe... */
05118         FILE *in;
05119         if (!(in = fopen(ctl->atm_gpfile, "r")))
05120             ERRMSG("Cannot open file!");
05121         char line[LEN];
05122         while (fgets(line, LEN, in))
05123             fprintf(out, "%s", line);
05124         fclose(in);
05125     }
05126
05127     else {
05128
05129         /* Create file... */

```

```

05130     if (!(out = fopen(filename, "w")))
05131         ERRMSG("Cannot create file!");
05132 }
05133
05134 /* Write header... */
05135 fprintf(out,
05136         "# $1 = time [s]\n"
05137         "# $2 = altitude [km]\n"
05138         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05139 for (int iq = 0; iq < ctl->nq; iq++)
05140     fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
05141             ctl->qnt_unit[iq]);
05142 fprintf(out, "\n");
05143
05144 /* Write data... */
05145 for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
05146
05147     /* Check time... */
05148     if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05149         continue;
05150
05151     /* Write output... */
05152     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
05153             atm->lon[ip], atm->lat[ip]);
05154     for (int iq = 0; iq < ctl->nq; iq++) {
05155         fprintf(out, " ");
05156         if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05157             fprintf(out, ctl->qnt_format[iq], GSL_NAN);
05158         else
05159             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05160     }
05161     fprintf(out, "\n");
05162 }
05163
05164 /* Close file... */
05165 fclose(out);
05166 }

```

Here is the call graph for this function:



**5.19.2.67 write\_atm\_bin()** void write\_atm\_bin (  
     const char \* filename,  
     ctl\_t \* ctl,  
     atm\_t \* atm )

Write atmospheric data in binary format.

Definition at line 5170 of file libtrac.c.

```

05173     {
05174
05175     FILE *out;
05176
05177     /* Create file... */
05178     if (!(out = fopen(filename, "w")))
05179         ERRMSG("Cannot create file!");
05180
05181     /* Write version of binary data... */
05182     int version = 100;
05183     FWRITE(&version, int,
05184           1,

```

```

05185         out);
05186
05187     /* Write data... */
05188     FWRITE(&atm->np, int,
05189           1,
05190           out);
05191     FWRITE(atm->time, double,
05192           (size_t) atm->np,
05193           out);
05194     FWRITE(atm->p, double,
05195           (size_t) atm->np,
05196           out);
05197     FWRITE(atm->lon, double,
05198           (size_t) atm->np,
05199           out);
05200     FWRITE(atm->lat, double,
05201           (size_t) atm->np,
05202           out);
05203     for (int iq = 0; iq < ctl->nq; iq++)
05204         FWRITE(atm->q[iq], double,
05205               (size_t) atm->np,
05206               out);
05207
05208     /* Write final flag... */
05209     int final = 999;
05210     FWRITE(&final, int,
05211           1,
05212           out);
05213
05214     /* Close file... */
05215     fclose(out);
05216 }

```

**5.19.2.68 write\_atm\_clams()** void write\_atm\_clams (

```

    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write atmospheric data in CLaMS format.

Definition at line 5220 of file libtrac.c.

```

05223     {
05224
05225     /* Global Counter... */
05226     static size_t out_cnt = 0;
05227
05228     char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";
05229
05230     double r, r_start, r_stop;
05231
05232     int year, mon, day, hour, min, sec;
05233     int year_start, mon_start, day_start, hour_start, min_start, sec_start;
05234     int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
05235     int ncid, varid, tid, pid, cid, zid, dim_ids[2];
05236
05237     /* time, nparc */
05238     size_t start[2], count[2];
05239
05240     /* Determine start and stop times of calculation... */
05241     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05242     jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
05243             &min_start, &sec_start, &r_start);
05244     jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
05245             &min_stop, &sec_stop, &r_stop);
05246
05247     /* Set filename... */
05248     sprintf(filename_out,
05249           "/traj_fix_3d_%02d%02d%02d%02d%02d%02d%02d.nc",
05250           year_start % 100, mon_start, day_start, hour_start,
05251           year_stop % 100, mon_stop, day_stop, hour_stop);
05252     printf("Write traj file: %s\n", filename_out);
05253
05254     /* Define hyperslap for the traj_file... */
05255     start[0] = out_cnt;
05256     start[1] = 0;
05257     count[0] = 1;
05258     count[1] = (size_t) atm->np;
05259

```



```

05260  /* Create the file at the first timestep... */
05261  if (out_cnt == 0) {
05262
05263      /* Create file... */
05264      nc_create(filename_out, NC_CLOBBER, &ncid);
05265
05266      /* Define dimensions... */
05267      NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
05268      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05269      NC(nc_def_dim(ncid, "TMDT", 7, &cid));
05270      dim_ids[0] = tid;
05271      dim_ids[1] = pid;
05272
05273      /* Define variables and their attributes... */
05274      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05275                "seconds since 2000-01-01 00:00:00 UTC");
05276      NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
05277      NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
05278      NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
05279      NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
05280      for (int iq = 0; iq < ctl->nq; iq++)
05281          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05282                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05283
05284      /* Define global attributes... */
05285      NC_PUT_ATT("exp_VERTCOORD_name", "zeta");
05286      NC_PUT_ATT("model", "MPTRAC");
05287
05288      /* End definitions... */
05289      NC(nc_enddef(ncid));
05290      NC(nc_close(ncid));
05291  }
05292
05293  /* Increment global counter to change hyperslap... */
05294  out_cnt++;
05295
05296  /* Open file... */
05297  NC(nc_open(filename_out, NC_WRITE, &ncid));
05298
05299  /* Write data... */
05300  NC_PUT_DOUBLE("time", atm->time, 1);
05301  NC_PUT_DOUBLE("LAT", atm->lat, 1);
05302  NC_PUT_DOUBLE("LON", atm->lon, 1);
05303  NC_PUT_DOUBLE("PRESS", atm->p, 1);
05304  if (ctl->vert_coord_ap == 1) {
05305      NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
05306  } else if (ctl->qnt_zeta >= 0) {
05307      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
05308  }
05309  for (int iq = 0; iq < ctl->nq; iq++)
05310      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);
05311
05312  /* Close file... */
05313  NC(nc_close(ncid));
05314
05315  /* At the last time step create the init_fix_YYYYMMDDHH file... */
05316  if ((year == year_stop) && (mon == mon_stop)
05317      && (day == day_stop) && (hour == hour_stop)) {
05318
05319      /* Set filename... */
05320      char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
05321      sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
05322              year_stop % 100, mon_stop, day_stop, hour_stop);
05323      printf("Write init file: %s\n", filename_init);
05324
05325      /* Create file... */
05326      nc_create(filename_init, NC_CLOBBER, &ncid);
05327
05328      /* Define dimensions... */
05329      NC(nc_def_dim(ncid, "time", 1, &tid));
05330      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05331      dim_ids[0] = tid;
05332      dim_ids[1] = pid;
05333
05334      /* Define variables and their attributes... */
05335      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05336                "seconds since 2000-01-01 00:00:00 UTC");
05337      NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
05338      NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
05339      NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
05340      NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
05341      NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
05342      NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
05343      for (int iq = 0; iq < ctl->nq; iq++)
05344          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05345                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05346

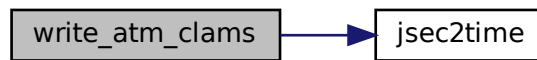
```

```

05347      /* Define global attributes... */
05348      NC_PUT_ATT("exp_VERTCOOR_name", "zeta");
05349      NC_PUT_ATT("model", "MPTRAC");
05350
05351      /* End definitions... */
05352      NC(nc_enddef(ncid));
05353
05354      /* Write data... */
05355      NC_PUT_DOUBLE("time", atm->time, 0);
05356      NC_PUT_DOUBLE("LAT", atm->lat, 0);
05357      NC_PUT_DOUBLE("LON", atm->lon, 0);
05358      NC_PUT_DOUBLE("PRESS", atm->p, 0);
05359      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
05360      for (int iq = 0; iq < ctl->nq; iq++)
05361          NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05362
05363      /* Close file... */
05364      NC(nc_close(ncid));
05365  }
05366 }

```

Here is the call graph for this function:



**5.19.2.69 write\_atm\_nc()** void write\_atm\_nc (

```

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

```

Write atmospheric data in netCDF format.

Definition at line 5370 of file libtrac.c.

```

05373      {
05374
05375      int ncid, obsid, varid;
05376
05377      size_t start[2], count[2];
05378
05379      /* Create file... */
05380      nc_create(filename, NC_CLOBBER, &ncid);
05381
05382      /* Define dimensions... */
05383      NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));
05384
05385      /* Define variables and their attributes... */
05386      NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
05387                "seconds since 2000-01-01 00:00:00 UTC");
05388      NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
05389      NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
05390      NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
05391      for (int iq = 0; iq < ctl->nq; iq++)
05392          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
05393                    ctl->qnt_longname[iq], ctl->qnt_unit[iq]);
05394
05395      /* Define global attributes... */
05396      NC_PUT_ATT("featureType", "point");
05397
05398      /* End definitions... */
05399      NC(nc_enddef(ncid));
05400
05401      /* Write data... */

```

```

05402 NC_PUT_DOUBLE("time", atm->time, 0);
05403 NC_PUT_DOUBLE("press", atm->p, 0);
05404 NC_PUT_DOUBLE("lon", atm->lon, 0);
05405 NC_PUT_DOUBLE("lat", atm->lat, 0);
05406 for (int iq = 0; iq < ctl->nq; iq++)
05407     NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05408
05409 /* Close file... */
05410 NC(nc_close(ncid));
05411 }

```

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

Write CSI data.

Definition at line 5415 of file libtrac.c.

```

05419 {
05420
05421     static FILE *out;
05422
05423     static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
05424         dlon, dlat, dz, x[NCSI], y[NCSI];
05425
05426     static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;
05427
05428     /* Set timer... */
05429     SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
05430
05431     /* Init... */
05432     if (t == ctl->t_start) {
05433
05434         /* Check quantity index for mass... */
05435         if (ctl->qnt_m < 0)
05436             ERRMSG("Need quantity mass!");
05437
05438         /* Allocate... */
05439         ALLOC(area, double,
05440             ctl->csi_ry);
05441         ALLOC(rt, double,
05442             NOBS);
05443         ALLOC(rz, double,
05444             NOBS);
05445         ALLOC(rlon, double,
05446             NOBS);
05447         ALLOC(rlat, double,
05448             NOBS);
05449         ALLOC(robs, double,
05450             NOBS);
05451
05452         /* Read observation data... */
05453         read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);
05454
05455         /* Create new file... */
05456         LOG(1, "Write CSI data: %s", filename);
05457         if (!(out = fopen(filename, "w")))
05458             ERRMSG("Cannot create file!");
05459
05460         /* Write header... */
05461         fprintf(out,
05462             "# $1 = time [s]\n"
05463             "# $2 = number of hits (cx)\n"
05464             "# $3 = number of misses (cy)\n"
05465             "# $4 = number of false alarms (cz)\n"
05466             "# $5 = number of observations (cx + cy)\n"
05467             "# $6 = number of forecasts (cx + cz)\n"
05468             "# $7 = bias (ratio of forecasts and observations) [%%]\n"
05469             "# $8 = probability of detection (POD) [%%]\n"
05470             "# $9 = false alarm rate (FAR) [%%]\n"
05471             "# $10 = critical success index (CSI) [%%]\n");
05472         fprintf(out,
05473             "# $11 = hits associated with random chance\n"
05474             "# $12 = equitable threat score (ETS) [%%]\n"
05475             "# $13 = Pearson linear correlation coefficient\n"

```

```

05476         "# $14 = Spearman rank-order correlation coefficient\n"
05477         "# $15 = column density mean error (F - O) [kg/m^2]\n"
05478         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
05479         "# $17 = column density mean absolute error [kg/m^2]\n"
05480         "# $18 = number of data points\n\n");
05481
05482     /* Set grid box size... */
05483     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
05484     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
05485     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
05486
05487     /* Set horizontal coordinates... */
05488     for (iy = 0; iy < ctl->csi_ny; iy++) {
05489         double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
05490         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
05491     }
05492 }
05493
05494 /* Set time interval... */
05495 double t0 = t - 0.5 * ctl->dt_mod;
05496 double t1 = t + 0.5 * ctl->dt_mod;
05497
05498 /* Allocate... */
05499 ALLOC(modmean, double,
05500       ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05501 ALLOC(obsmean, double,
05502       ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05503 ALLOC(obscount, int,
05504       ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05505
05506 /* Loop over observations... */
05507 for (int i = 0; i < nob; i++) {
05508
05509     /* Check time... */
05510     if (rt[i] < t0)
05511         continue;
05512     else if (rt[i] >= t1)
05513         break;
05514
05515     /* Check observation data... */
05516     if (!isfinite(robs[i]))
05517         continue;
05518
05519     /* Calculate indices... */
05520     ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
05521     iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
05522     iz = (int) ((rz[i] - ctl->csi_z0) / dz);
05523
05524     /* Check indices... */
05525     if (ix < 0 || ix >= ctl->csi_nx ||
05526         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05527         continue;
05528
05529     /* Get mean observation index... */
05530     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05531     obsmean[idx] += robs[i];
05532     obscount[idx]++;
05533 }
05534
05535 /* Analyze model data... */
05536 for (ip = 0; ip < atm->np; ip++) {
05537
05538     /* Check time... */
05539     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05540         continue;
05541
05542     /* Get indices... */
05543     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
05544     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
05545     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
05546
05547     /* Check indices... */
05548     if (ix < 0 || ix >= ctl->csi_nx ||
05549         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05550         continue;
05551
05552     /* Get total mass in grid cell... */
05553     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05554     modmean[idx] += atm->q[ctl->qnt_m][ip];
05555 }
05556
05557 /* Analyze all grid cells... */
05558 for (ix = 0; ix < ctl->csi_nx; ix++)
05559     for (iy = 0; iy < ctl->csi_ny; iy++)
05560         for (iz = 0; iz < ctl->csi_nz; iz++) {
05561
05562             /* Calculate mean observation index... */

```

```

05563     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05564     if (obscount[idx] > 0)
05565         obsmean[idx] /= obscount[idx];
05566
05567     /* Calculate column density... */
05568     if (modmean[idx] > 0)
05569         modmean[idx] /= (1e6 * area[iy]);
05570
05571     /* Calculate CSI... */
05572     if (obscount[idx] > 0) {
05573         ct++;
05574         if (obsmean[idx] >= ctl->csi_obsmin &&
05575             modmean[idx] >= ctl->csi_modmin)
05576             cx++;
05577         else if (obsmean[idx] >= ctl->csi_obsmin &&
05578                 modmean[idx] < ctl->csi_modmin)
05579             cy++;
05580         else if (obsmean[idx] < ctl->csi_obsmin &&
05581                 modmean[idx] >= ctl->csi_modmin)
05582             cz++;
05583     }
05584
05585     /* Save data for other verification statistics... */
05586     if (obscount[idx] > 0
05587         && (obsmean[idx] >= ctl->csi_obsmin
05588             || modmean[idx] >= ctl->csi_modmin)) {
05589         x[n] = modmean[idx];
05590         y[n] = obsmean[idx];
05591         if (++n > NCSI)
05592             ERRMSG("Too many data points to calculate statistics!");
05593     }
05594 }
05595
05596 /* Write output... */
05597 if (fmod(t, ctl->csi_dt_out) == 0) {
05598
05599     /* Calculate verification statistics
05600      (https://www.cawcr.gov.au/projects/verification/) ... */
05601     static double work[2 * NCSI];
05602     int n_obs = cx + cy;
05603     int n_for = cx + cz;
05604     double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
05605     double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
05606     double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
05607     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
05608     double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
05609     double ets = (cx + cy + cz - cx_rd > 0) ?
05610         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
05611     double rho_p =
05612         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
05613     double rho_s =
05614         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
05615     for (int i = 0; i < n; i++)
05616         work[i] = x[i] - y[i];
05617     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
05618     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
05619         0.0) : GSL_NAN;
05620     double absdev =
05621         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
05622
05623     /* Write... */
05624     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %d\n",
05625         t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
05626         rho_p, rho_s, mean, rmse, absdev, n);
05627
05628     /* Set counters to zero... */
05629     n = ct = cx = cy = cz = 0;
05630 }
05631
05632 /* Free... */
05633 free(modmean);
05634 free(obsmean);
05635 free(obscount);
05636
05637 /* Finalize... */
05638 if (t == ctl->t_stop) {
05639
05640     /* Close output file... */
05641     fclose(out);
05642
05643     /* Free... */
05644     free(area);
05645     free(rt);
05646     free(rz);
05647     free(rlon);
05648     free(rlat);
05649     free(robs);

```

```
05650     }
05651 }
```

Here is the call graph for this function:



**5.19.2.71 write\_ens()** void write\_ens (

```

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

Write ensemble data.

Definition at line 5655 of file libtrac.c.

```

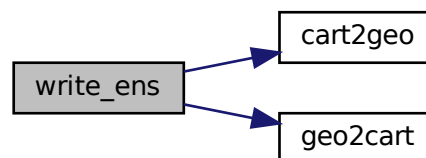
05659     {
05660
05661     static FILE *out;
05662
05663     static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
05664         x[3], zm[NENS];
05665
05666     static int n[NENS];
05667
05668     /* Set timer... */
05669     SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
05670
05671     /* Check quantities... */
05672     if (ctl->qnt_ens < 0)
05673         ERRMSG("Missing ensemble IDs!");
05674
05675     /* Set time interval... */
05676     double t0 = t - 0.5 * ctl->dt_mod;
05677     double t1 = t + 0.5 * ctl->dt_mod;
05678
05679     /* Init... */
05680     for (int i = 0; i < NENS; i++) {
05681         for (int iq = 0; iq < ctl->nq; iq++)
05682             qm[iq][i] = qs[iq][i] = 0;
05683         xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
05684         n[i] = 0;
05685     }
05686
05687     /* Loop over air parcels... */
05688     for (int ip = 0; ip < atm->np; ip++) {
05689
05690         /* Check time... */
05691         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05692             continue;
05693
05694         /* Check ensemble ID... */
05695         if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
05696             ERRMSG("Ensemble ID is out of range!");
05697
05698         /* Get means... */
05699         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
05700         for (int iq = 0; iq < ctl->nq; iq++) {
05701             qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
05702             qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
05703         }
05704         xm[ctl->qnt_ens][0] += x[0];
```

```

05705     xm[ctl->qnt_ens][1] += x[1];
05706     xm[ctl->qnt_ens][2] += x[2];
05707     zm[ctl->qnt_ens] += Z(atm->p[ip]);
05708     n[ctl->qnt_ens]++;
05709 }
05710
05711 /* Create file... */
05712 LOG(1, "Write ensemble data: %s", filename);
05713 if (!(out = fopen(filename, "w")))
05714     ERRMSG("Cannot create file!");
05715
05716 /* Write header... */
05717 fprintf(out,
05718         "# $1 = time [s]\n"
05719         "# $2 = altitude [km]\n"
05720         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05721 for (int iq = 0; iq < ctl->nq; iq++)
05722     fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
05723             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05724 for (int iq = 0; iq < ctl->nq; iq++)
05725     fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
05726             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05727 fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
05728
05729 /* Write data... */
05730 for (int i = 0; i < NENS; i++)
05731     if (n[i] > 0) {
05732         cart2geo(xm[i], &dummy, &lon, &lat);
05733         fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
05734         for (int iq = 0; iq < ctl->nq; iq++) {
05735             fprintf(out, " ");
05736             fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
05737         }
05738         for (int iq = 0; iq < ctl->nq; iq++) {
05739             fprintf(out, " ");
05740             double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
05741             fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
05742         }
05743         fprintf(out, " %d\n", n[i]);
05744     }
05745
05746 /* Close file... */
05747 fclose(out);
05748 }

```

Here is the call graph for this function:



**5.19.2.72 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 5752 of file libtrac.c.

```

05758     {
05759
05760     double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;
05761
05762     int *ixs, *iys, *izs, *np;
05763
05764     /* Set timer... */
05765     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
05766
05767     /* Write info... */
05768     LOG(1, "Write grid data: %s", filename);
05769
05770     /* Allocate... */
05771     ALLOC(cd, double,
05772           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05773     ALLOC(mass, double,
05774           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05775     ALLOC(vmr_expl, double,
05776           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05777     ALLOC(vmr_impl, double,
05778           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05779     ALLOC(z, double,
05780           ctl->grid_nz);
05781     ALLOC(lon, double,
05782           ctl->grid_nx);
05783     ALLOC(lat, double,
05784           ctl->grid_ny);
05785     ALLOC(area, double,
05786           ctl->grid_ny);
05787     ALLOC(press, double,
05788           ctl->grid_nz);
05789     ALLOC(np, int,
05790           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05791     ALLOC(ixs, int,
05792           atm->np);
05793     ALLOC(iys, int,
05794           atm->np);
05795     ALLOC(izs, int,
05796           atm->np);
05797
05798     /* Set grid box size... */
05799     double dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
05800     double dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
05801     double dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
05802
05803     /* Set vertical coordinates... */
05804     #pragma omp parallel for default(shared)
05805     for (int iz = 0; iz < ctl->grid_nz; iz++) {
05806         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
05807         press[iz] = P(z[iz]);
05808     }
05809
05810     /* Set horizontal coordinates... */
05811     for (int ix = 0; ix < ctl->grid_nx; ix++)
05812         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
05813     #pragma omp parallel for default(shared)
05814     for (int iy = 0; iy < ctl->grid_ny; iy++) {
05815         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
05816         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05817             * cos(lat[iy] * M_PI / 180.);
05818     }
05819
05820     /* Set time interval for output... */
05821     double t0 = t - 0.5 * ctl->dt_mod;
05822     double t1 = t + 0.5 * ctl->dt_mod;
05823
05824     /* Get grid box indices... */
05825     #pragma omp parallel for default(shared)
05826     for (int ip = 0; ip < atm->np; ip++) {
05827         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
05828         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
05829         izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05830         if (atm->time[ip] < t0 || atm->time[ip] > t1
05831             || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05832             || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05833             || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05834             izs[ip] = -1;
05835     }
05836
05837     /* Average data... */
05838     for (int ip = 0; ip < atm->np; ip++)
05839         if (izs[ip] >= 0) {
05840             int idx =
05841                 ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
05842             np[idx]++;
05843             if (ctl->qnt_m >= 0)

```



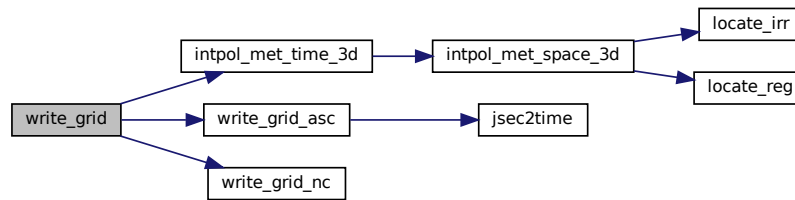
```

05844     mass[idx] += atm->q[ctl->qnt_m][ip];
05845     if (ctl->qnt_vmr >= 0)
05846         vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
05847 }
05848
05849 /* Get implicit vmr per particle... */
05850 if (ctl->qnt_vmrimpl >= 0)
05851     for (int ip = 0; ip < atm->np; ip++)
05852         if (izs[ip] >= 0) {
05853             double temp;
05854             INTPOL_INIT;
05855             intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
05856                             lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
05857             atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass
05858                 *
05859                 mass[ARRAY_3D
05860                     (ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz)]
05861                 / (RHO(press[izs[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
05862         }
05863
05864 /* Calculate column density and vmr... */
05865 #pragma omp parallel for default(shared)
05866 for (int ix = 0; ix < ctl->grid_nx; ix++)
05867     for (int iy = 0; iy < ctl->grid_ny; iy++)
05868         for (int iz = 0; iz < ctl->grid_nz; iz++) {
05869
05870             /* Get grid index... */
05871             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
05872
05873             /* Calculate column density... */
05874             cd[idx] = GSL_NAN;
05875             if (ctl->qnt_m >= 0)
05876                 cd[idx] = mass[idx] / (1e6 * area[iy]);
05877
05878             /* Calculate volume mixing ratio (implicit)... */
05879             vmr_impl[idx] = GSL_NAN;
05880             if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05881                 vmr_impl[idx] = 0;
05882                 if (mass[idx] > 0) {
05883
05884                     /* Get temperature... */
05885                     double temp;
05886                     INTPOL_INIT;
05887                     intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05888                                     lon[ix], lat[iy], &temp, ci, cw, 1);
05889
05890                     /* Calculate volume mixing ratio... */
05891                     vmr_impl[idx] = MA / ctl->molmass * mass[idx]
05892                                     / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
05893                 }
05894             }
05895
05896             /* Calculate volume mixing ratio (explicit)... */
05897             if (ctl->qnt_vmr >= 0 && np[idx] > 0)
05898                 vmr_expl[idx] /= np[idx];
05899             else
05900                 vmr_expl[idx] = GSL_NAN;
05901         }
05902
05903 /* Write ASCII data... */
05904 if (ctl->grid_type == 0)
05905     write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
05906                  t, z, lon, lat, area, dz, np);
05907
05908 /* Write netCDF data... */
05909 else if (ctl->grid_type == 1)
05910     write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
05911                  t, z, lon, lat, area, dz, np);
05912
05913 /* Error message... */
05914 else
05915     ERRMSG("Grid data format GRID_TYPE unknown!");
05916
05917 /* Free... */
05918 free(cd);
05919 free(mass);
05920 free(vmr_expl);
05921 free(vmr_impl);
05922 free(z);
05923 free(lon);
05924 free(lat);
05925 free(area);
05926 free(press);
05927 free(np);
05928 free(ixs);
05929 free(iys);
05930 free(izs);

```

```
05931 }
```

Here is the call graph for this function:



```

5.19.2.73 write_grid_asc() void write_grid_asc (
    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )

```

Write gridded data in ASCII format.

Definition at line 5935 of file libtrac.c.

```

05947     {
05948
05949     FILE *in, *out;
05950
05951     char line[LEN];
05952
05953     /* Check if gnuplot output is requested... */
05954     if (ctl->grid_gpfile[0] != '-') {
05955
05956         /* Create gnuplot pipe... */
05957         if (!(out = popen("gnuplot", "w")))
05958             ERRMSG("Cannot create pipe to gnuplot!");
05959
05960         /* Set plot filename... */
05961         fprintf(out, "set out \"%s.png\"\n", filename);
05962
05963         /* Set time string... */
05964         double r;
05965         int year, mon, day, hour, min, sec;
05966         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05967         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05968             year, mon, day, hour, min);
05969
05970         /* Dump gnuplot file to pipe... */
05971         if (!(in = fopen(ctl->grid_gpfile, "r")))
05972             ERRMSG("Cannot open file!");
05973         while (fgets(line, LEN, in))
05974             fprintf(out, "%s", line);
05975         fclose(in);
05976     }
05977
05978     else {

```

```

05979
05980     /* Create file... */
05981     if (!out = fopen(filename, "w"))
05982         ERRMSG("Cannot create file!");
05983 }
05984
05985 /* Write header... */
05986 fprintf(out,
05987     "# $1 = time [s]\n"
05988     "# $2 = altitude [km]\n"
05989     "# $3 = longitude [deg]\n"
05990     "# $4 = latitude [deg]\n"
05991     "# $5 = surface area [km^2]\n"
05992     "# $6 = layer depth [km]\n"
05993     "# $7 = number of particles [l]\n"
05994     "# $8 = column density (implicit) [kg/m^2]\n"
05995     "# $9 = volume mixing ratio (implicit) [ppv]\n"
05996     "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
05997
05998 /* Write data... */
05999 for (int ix = 0; ix < ctl->grid_nx; ix++) {
06000     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
06001         fprintf(out, "\n");
06002     for (int iy = 0; iy < ctl->grid_ny; iy++) {
06003         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
06004             fprintf(out, "\n");
06005         for (int iz = 0; iz < ctl->grid_nz; iz++) {
06006             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
06007             if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
06008                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
06009                     t, z[iz], lon[ix], lat[iy], area[iy], dz,
06010                     np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
06011         }
06012     }
06013 }
06014
06015 /* Close file... */
06016 fclose(out);
06017 }

```

Here is the call graph for this function:



**5.19.2.74 write\_grid\_nc()** void write\_grid\_nc (

```

    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )

```

Write gridded data in netCDF format.

Definition at line 6021 of file libtrac.c.

```

06033     {
06034
06035     int ncid, dimid[10], varid;
06036
06037     size_t start[2], count[2];
06038
06039     /* Create file... */
06040     nc_create(filename, NC_CLOBBER, &ncid);
06041
06042     /* Define dimensions... */
06043     NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
06044     NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[1]));
06045     NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
06046     NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[3]));
06047     NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));
06048
06049     /* Define variables and their attributes... */
06050     NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
06051              "seconds since 2000-01-01 00:00:00 UTC");
06052     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[1], "longitude", "degrees_east");
06053     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
06054     NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[3], "altitude", "km");
06055     NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
06056     NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[4], "layer depth", "km");
06057     NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
06058     NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
06059              "volume mixing ratio (implicit)", "ppv");
06060     NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
06061              "volume mixing ratio (explicit)", "ppv");
06062     NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");
06063
06064     /* End definitions... */
06065     NC(nc_enddef(ncid));
06066
06067     /* Write data... */
06068     NC_PUT_DOUBLE("time", &t, 0);
06069     NC_PUT_DOUBLE("lon", lon, 0);
06070     NC_PUT_DOUBLE("lat", lat, 0);
06071     NC_PUT_DOUBLE("z", z, 0);
06072     NC_PUT_DOUBLE("area", area, 0);
06073     NC_PUT_DOUBLE("dz", &dz, 0);
06074     NC_PUT_DOUBLE("cd", cd, 0);
06075     NC_PUT_DOUBLE("vmr_impl", vmr_impl, 0);
06076     NC_PUT_DOUBLE("vmr_expl", vmr_expl, 0);
06077     NC_PUT_INT("np", np, 0);
06078
06079     /* Close file... */
06080     NC(nc_close(ncid));
06081 }

```

```

5.19.2.75 write_met() int write_met (
    char * filename,
    ctl_t * ctl,
    met_t * met )

```

Read meteo data file.

Definition at line 6085 of file libtrac.c.

```

06088     {
06089
06090     /* Set timer... */
06091     SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);
06092
06093     /* Write info... */
06094     LOG(1, "Write meteo data: %s", filename);
06095
06096     /* Check compression flags... */
06097     #ifndef ZFP
06098     if (ctl->met_type == 3)
06099         ERRMSG("zfp compression not supported!");
06100     #endif
06101     #ifndef ZSTD
06102     if (ctl->met_type == 4)
06103         ERRMSG("zstd compression not supported!");
06104     #endif
06105
06106     /* Write binary... */

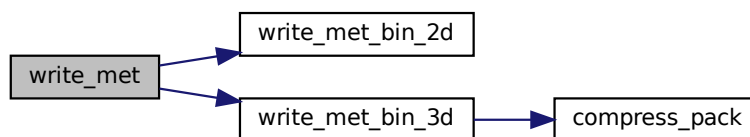
```

```

06107     if (ctl->met_type >= 1 && ctl->met_type <= 4) {
06108
06109         /* Create file... */
06110         FILE *out;
06111         if (!(out = fopen(filename, "w")))
06112             ERRMSG("Cannot create file!");
06113
06114         /* Write type of binary data... */
06115         FWRITE(&ctl->met_type, int,
06116             1,
06117             out);
06118
06119         /* Write version of binary data... */
06120         int version = 100;
06121         FWRITE(&version, int,
06122             1,
06123             out);
06124
06125         /* Write grid data... */
06126         FWRITE(&met->time, double,
06127             1,
06128             out);
06129         FWRITE(&met->nx, int,
06130             1,
06131             out);
06132         FWRITE(&met->ny, int,
06133             1,
06134             out);
06135         FWRITE(&met->np, int,
06136             1,
06137             out);
06138         FWRITE(met->lon, double,
06139             (size_t) met->nx,
06140             out);
06141         FWRITE(met->lat, double,
06142             (size_t) met->ny,
06143             out);
06144         FWRITE(met->p, double,
06145             (size_t) met->np,
06146             out);
06147
06148         /* Write surface data... */
06149         write_met_bin_2d(out, met, met->ps, "PS");
06150         write_met_bin_2d(out, met, met->ts, "TS");
06151         write_met_bin_2d(out, met, met->zs, "ZS");
06152         write_met_bin_2d(out, met, met->us, "US");
06153         write_met_bin_2d(out, met, met->vs, "VS");
06154         write_met_bin_2d(out, met, met->pbl, "PBL");
06155         write_met_bin_2d(out, met, met->pt, "PT");
06156         write_met_bin_2d(out, met, met->tt, "TT");
06157         write_met_bin_2d(out, met, met->zt, "ZT");
06158         write_met_bin_2d(out, met, met->h2ot, "H2OT");
06159         write_met_bin_2d(out, met, met->pct, "PCT");
06160         write_met_bin_2d(out, met, met->pcb, "PCB");
06161         write_met_bin_2d(out, met, met->cl, "CL");
06162         write_met_bin_2d(out, met, met->plcl, "PLCL");
06163         write_met_bin_2d(out, met, met->plfc, "PLFC");
06164         write_met_bin_2d(out, met, met->pel, "PEL");
06165         write_met_bin_2d(out, met, met->cape, "CAPE");
06166         write_met_bin_2d(out, met, met->cin, "CIN");
06167
06168         /* Write level data... */
06169         write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
06170         write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
06171         write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
06172         write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
06173         write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
06174         write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
06175         write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
06176         write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
06177         write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
06178         write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);
06179
06180         /* Write final flag... */
06181         int final = 999;
06182         FWRITE(&final, int,
06183             1,
06184             out);
06185
06186         /* Close file... */
06187         fclose(out);
06188     }
06189
06190     return 0;
06191 }

```

Here is the call graph for this function:



**5.19.2.76 write\_met\_bin\_2d()** void write\_met\_bin\_2d (  
 FILE \* out,  
 met\_t \* met,  
 float var[EX][EY],  
 char \* varname )

Write 2-D meteo variable.

Definition at line 6195 of file libtrac.c.

```

06199     {
06200
06201     float *help;
06202
06203     /* Allocate... */
06204     ALLOC(help, float,
06205           EX * EY);
06206
06207     /* Copy data... */
06208     for (int ix = 0; ix < met->nx; ix++)
06209         for (int iy = 0; iy < met->ny; iy++)
06210             help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];
06211
06212     /* Write uncompressed data... */
06213     LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
06214     FWRITE(help, float,
06215            (size_t) (met->nx * met->ny),
06216            out);
06217
06218     /* Free... */
06219     free(help);
06220 }
  
```

**5.19.2.77 write\_met\_bin\_3d()** void write\_met\_bin\_3d (  
 FILE \* out,  
 ctl\_t \* ctl,  
 met\_t \* met,  
 float var[EX][EY][EP],  
 char \* varname,  
 int precision,  
 double tolerance )

Write 3-D meteo variable.

Definition at line 6224 of file libtrac.c.

```

06231     {
  
```

```

06232
06233     float *help;
06234
06235     /* Allocate... */
06236     ALLOC(help, float,
06237           EX * EY * EP);
06238
06239     /* Copy data... */
06240     #pragma omp parallel for default(shared) collapse(2)
06241     for (int ix = 0; ix < met->nx; ix++)
06242         for (int iy = 0; iy < met->ny; iy++)
06243             for (int ip = 0; ip < met->np; ip++)
06244                 help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];
06245
06246     /* Write uncompressed data... */
06247     if (ctl->met_type == 1) {
06248         LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
06249         FWRITE(help, float,
06250               (size_t) (met->nx * met->ny * met->np),
06251               out);
06252     }
06253
06254     /* Write packed data... */
06255     else if (ctl->met_type == 2)
06256         compress_pack(varname, help, (size_t) (met->ny * met->nx),
06257                       (size_t) met->np, 0, out);
06258
06259     /* Write zfp data... */
06260     #ifndef ZFP
06261     else if (ctl->met_type == 3)
06262         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
06263                     tolerance, 0, out);
06264     #endif
06265
06266     /* Write zstd data... */
06267     #ifndef ZSTD
06268     else if (ctl->met_type == 4)
06269         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
06270                      out);
06271     #endif
06272
06273     /* Unknown method... */
06274     else {
06275         ERRMSG("MET_TYPE not supported!");
06276         LOG(3, "%d %g", precision, tolerance);
06277     }
06278
06279     /* Free... */
06280     free(help);
06281 }

```

Here is the call graph for this function:



**5.19.2.78 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 6285 of file libtrac.c.

```

6291     {
6292
6293     static FILE *out;
6294
6295     static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
6296         dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;
6297
6298     static int nob, *obscount, ip, okay;
6299
6300     /* Set timer... */
6301     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
6302
6303     /* Init... */
6304     if (t == ctl->t_start) {
6305
6306         /* Check quantity index for mass... */
6307         if (ctl->qnt_m < 0)
6308             ERRMSG("Need quantity mass!");
6309
6310         /* Check molar mass... */
6311         if (ctl->molmass <= 0)
6312             ERRMSG("Specify molar mass!");
6313
6314         /* Allocate... */
6315         ALLOC(lon, double,
6316             ctl->prof_nx);
6317         ALLOC(lat, double,
6318             ctl->prof_ny);
6319         ALLOC(area, double,
6320             ctl->prof_ny);
6321         ALLOC(z, double,
6322             ctl->prof_nz);
6323         ALLOC(press, double,
6324             ctl->prof_nz);
6325         ALLOC(rt, double,
6326             NOBS);
6327         ALLOC(rz, double,
6328             NOBS);
6329         ALLOC(rlon, double,
6330             NOBS);
6331         ALLOC(rlat, double,
6332             NOBS);
6333         ALLOC(robs, double,
6334             NOBS);
6335
6336         /* Read observation data... */
6337         read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nob);
6338
6339         /* Create new output file... */
6340         LOG(1, "Write profile data: %s", filename);
6341         if (!(out = fopen(filename, "w")))
6342             ERRMSG("Cannot create file!");
6343
6344         /* Write header... */
6345         fprintf(out,
6346             "# $1 = time [s]\n"
6347             "# $2 = altitude [km]\n"
6348             "# $3 = longitude [deg]\n"
6349             "# $4 = latitude [deg]\n"
6350             "# $5 = pressure [hPa]\n"
6351             "# $6 = temperature [K]\n"
6352             "# $7 = volume mixing ratio [ppv]\n"
6353             "# $8 = H2O volume mixing ratio [ppv]\n"
6354             "# $9 = O3 volume mixing ratio [ppv]\n"
6355             "# $10 = observed BT index [K]\n"
6356             "# $11 = number of observations\n");
6357
6358         /* Set grid box size... */
6359         dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
6360         dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
6361         dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
6362
6363         /* Set vertical coordinates... */
6364         for (int iz = 0; iz < ctl->prof_nz; iz++) {
6365             z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
6366             press[iz] = P(z[iz]);
6367         }
6368
6369         /* Set horizontal coordinates... */
6370         for (int ix = 0; ix < ctl->prof_nx; ix++)
6371             lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
6372         for (int iy = 0; iy < ctl->prof_ny; iy++) {
6373             lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);

```



```

06374         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
06375         * cos(lat[iy] * M_PI / 180.);
06376     }
06377 }
06378
06379 /* Set time interval... */
06380 double t0 = t - 0.5 * ctl->dt_mod;
06381 double t1 = t + 0.5 * ctl->dt_mod;
06382
06383 /* Allocate... */
06384 ALLOC(mass, double,
06385       ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
06386 ALLOC(obsmean, double,
06387       ctl->prof_nx * ctl->prof_ny);
06388 ALLOC(obscount, int,
06389       ctl->prof_nx * ctl->prof_ny);
06390
06391 /* Loop over observations... */
06392 for (int i = 0; i < nob; i++) {
06393
06394     /* Check time... */
06395     if (rt[i] < t0)
06396         continue;
06397     else if (rt[i] >= t1)
06398         break;
06399
06400     /* Check observation data... */
06401     if (!isfinite(robs[i]))
06402         continue;
06403
06404     /* Calculate indices... */
06405     int ix = (int) ((rlon[i] - ctl->prof_lon0) / dlon);
06406     int iy = (int) ((rlat[i] - ctl->prof_lat0) / dlat);
06407
06408     /* Check indices... */
06409     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
06410         continue;
06411
06412     /* Get mean observation index... */
06413     int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
06414     obsmean[idx] += robs[i];
06415     obscount[idx]++;
06416 }
06417
06418 /* Analyze model data... */
06419 for (ip = 0; ip < atm->np; ip++) {
06420
06421     /* Check time... */
06422     if (atm->time[ip] < t0 || atm->time[ip] > t1)
06423         continue;
06424
06425     /* Get indices... */
06426     int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
06427     int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
06428     int iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
06429
06430     /* Check indices... */
06431     if (ix < 0 || ix >= ctl->prof_nx ||
06432         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
06433         continue;
06434
06435     /* Get total mass in grid cell... */
06436     int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06437     mass[idx] += atm->q[ctl->qnt_m][ip];
06438 }
06439
06440 /* Extract profiles... */
06441 for (int ix = 0; ix < ctl->prof_nx; ix++)
06442     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06443         int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
06444         if (obscount[idx2] > 0) {
06445
06446             /* Check profile... */
06447             okay = 0;
06448             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06449                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06450                 if (mass[idx3] > 0) {
06451                     okay = 1;
06452                     break;
06453                 }
06454             }
06455             if (!okay)
06456                 continue;
06457
06458             /* Write output... */
06459             fprintf(out, "\n");
06460

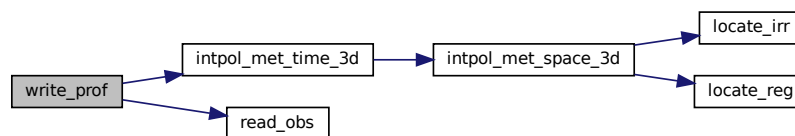
```

```

06461      /* Loop over altitudes... */
06462      for (int iz = 0; iz < ctl->prof_nz; iz++) {
06463
06464          /* Get temperature, water vapor, and ozone... */
06465          INTPOL_INIT;
06466          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
06467                          lon[ix], lat[iy], &temp, ci, cw, 1);
06468          intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
06469                          lon[ix], lat[iy], &h2o, ci, cw, 0);
06470          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
06471                          lon[ix], lat[iy], &o3, ci, cw, 0);
06472
06473          /* Calculate volume mixing ratio... */
06474          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06475          vmr = MA / ctl->molmass * mass[idx3]
06476              / (RHO(press[iz], temp) * area[iy] * dz * 1e9);
06477
06478          /* Write output... */
06479          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %d\n",
06480                  t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
06481                  obsmean[idx2] / obscount[idx2], obscount[idx2]);
06482      }
06483  }
06484  }
06485
06486  /* Free... */
06487  free(mass);
06488  free(obsmean);
06489  free(obscount);
06490
06491  /* Finalize... */
06492  if (t == ctl->t_stop) {
06493
06494      /* Close output file... */
06495      fclose(out);
06496
06497      /* Free... */
06498      free(lon);
06499      free(lat);
06500      free(area);
06501      free(z);
06502      free(press);
06503      free(rt);
06504      free(rz);
06505      free(rlon);
06506      free(rlat);
06507      free(robs);
06508  }
06509  }

```

Here is the call graph for this function:



**5.19.2.79 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 6513 of file libtrac.c.

```

06519     {
06520
06521     static FILE *out;
06522
06523     static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;
06524
06525     static int nob;
06526
06527     /* Set timer... */
06528     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
06529
06530     /* Init... */
06531     if (t == ctl->t_start) {
06532
06533         /* Allocate... */
06534         ALLOC(rt, double,
06535             NOBS);
06536         ALLOC(rz, double,
06537             NOBS);
06538         ALLOC(rlon, double,
06539             NOBS);
06540         ALLOC(rlat, double,
06541             NOBS);
06542         ALLOC(robs, double,
06543             NOBS);
06544
06545         /* Read observation data... */
06546         read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nob);
06547
06548         /* Create output file... */
06549         LOG(1, "Write sample data: %s", filename);
06550         if (!(out = fopen(filename, "w")))
06551             ERRMSG("Cannot create file!");
06552
06553         /* Write header... */
06554         fprintf(out,
06555             "# $1 = time [s]\n"
06556             "# $2 = altitude [km]\n"
06557             "# $3 = longitude [deg]\n"
06558             "# $4 = latitude [deg]\n"
06559             "# $5 = surface area [km^2]\n"
06560             "# $6 = layer depth [km]\n"
06561             "# $7 = number of particles [1]\n"
06562             "# $8 = column density [kg/m^2]\n"
06563             "# $9 = volume mixing ratio [ppv]\n"
06564             "# $10 = observed BT index [K]\n\n");
06565
06566         /* Set latitude range, squared radius, and area... */
06567         dlat = DY2DEG(ctl->sample_dx);
06568         rmax2 = SQR(ctl->sample_dx);
06569         area = M_PI * rmax2;
06570     }
06571
06572     /* Set time interval for output... */
06573     double t0 = t - 0.5 * ctl->dt_mod;
06574     double t1 = t + 0.5 * ctl->dt_mod;
06575
06576     /* Loop over observations... */
06577     for (int i = 0; i < nob; i++) {
06578
06579         /* Check time... */
06580         if (rt[i] < t0)
06581             continue;
06582         else if (rt[i] >= t1)
06583             break;
06584
06585         /* Calculate Cartesian coordinates... */
06586         double x0[3];
06587         geo2cart(0, rlon[i], rlat[i], x0);
06588
06589         /* Set pressure range... */
06590         double rp = P(rz[i]);
06591         double ptop = P(rz[i] + ctl->sample_dz);
06592         double pbot = P(rz[i] - ctl->sample_dz);
06593
06594         /* Init... */
06595         double mass = 0;
06596         int np = 0;
06597
06598         /* Loop over air parcels... */
06599 #pragma omp parallel for default(shared) reduction(+:mass,np)
06600         for (int ip = 0; ip < atm->np; ip++) {
06601

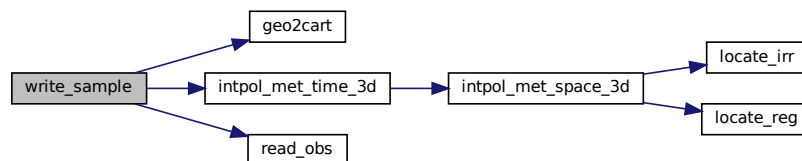
```

```

06602      /* Check time... */
06603      if (atm->time[ip] < t0 || atm->time[ip] > t1)
06604          continue;
06605
06606      /* Check latitude... */
06607      if (fabs(rlat[i] - atm->lat[ip]) > dlat)
06608          continue;
06609
06610      /* Check horizontal distance... */
06611      double x1[3];
06612      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06613      if (DIST2(x0, x1) > rmax2)
06614          continue;
06615
06616      /* Check pressure... */
06617      if (ctl->sample_dz > 0)
06618          if (atm->p[ip] > pbot || atm->p[ip] < ptop)
06619              continue;
06620
06621      /* Add mass... */
06622      if (ctl->qnt_m >= 0)
06623          mass += atm->q[ctl->qnt_m][ip];
06624      np++;
06625  }
06626
06627      /* Calculate column density... */
06628      double cd = mass / (1e6 * area);
06629
06630      /* Calculate volume mixing ratio... */
06631      double vmr = 0;
06632      if (ctl->molmass > 0 && ctl->sample_dz > 0) {
06633          if (mass > 0) {
06634
06635              /* Get temperature... */
06636              double temp;
06637              INTPOL_INIT;
06638              intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
06639                               rlon[i], rlat[i], &temp, ci, cw, 1);
06640
06641              /* Calculate volume mixing ratio... */
06642              vmr = MA / ctl->molmass * mass
06643                  / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
06644          }
06645      } else
06646          vmr = GSL_NAN;
06647
06648      /* Write output... */
06649      fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
06650              rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
06651  }
06652
06653      /* Finalize..... */
06654      if (t == ctl->t_stop) {
06655
06656          /* Close output file... */
06657          fclose(out);
06658
06659          /* Free... */
06660          free(rt);
06661          free(rz);
06662          free(rlon);
06663          free(rlat);
06664          free(robs);
06665      }
06666  }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 6670 of file libtrac.c.

```

06674     {
06675
06676     static FILE *out;
06677
06678     static double rmax2, x0[3], x1[3];
06679
06680     /* Set timer... */
06681     SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
06682
06683     /* Init... */
06684     if (t == ctl->t_start) {
06685
06686         /* Write info... */
06687         LOG(1, "Write station data: %s", filename);
06688
06689         /* Create new file... */
06690         if (!(out = fopen(filename, "w")))
06691             ERRMSG("Cannot create file!");
06692
06693         /* Write header... */
06694         fprintf(out,
06695             "# $1 = time [s]\n"
06696             "# $2 = altitude [km]\n"
06697             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
06698         for (int iq = 0; iq < ctl->nq; iq++)
06699             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
06700                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
06701         fprintf(out, "\n");
06702
06703         /* Set geolocation and search radius... */
06704         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
06705         rmax2 = SQR(ctl->stat_r);
06706     }
06707
06708     /* Set time interval for output... */
06709     double t0 = t - 0.5 * ctl->dt_mod;
06710     double t1 = t + 0.5 * ctl->dt_mod;
06711
06712     /* Loop over air parcels... */
06713     for (int ip = 0; ip < atm->np; ip++) {
06714
06715         /* Check time... */
06716         if (atm->time[ip] < t0 || atm->time[ip] > t1)
06717             continue;
06718
06719         /* Check time range for station output... */
06720         if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
06721             continue;
06722
06723         /* Check station flag... */
06724         if (ctl->qnt_stat >= 0)
06725             if (atm->q[ctl->qnt_stat][ip])
06726                 continue;
06727
06728         /* Get Cartesian coordinates... */
06729         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06730
06731         /* Check horizontal distance... */
06732         if (DIST2(x0, x1) > rmax2)
06733             continue;
06734
06735         /* Set station flag... */
06736         if (ctl->qnt_stat >= 0)
06737             atm->q[ctl->qnt_stat][ip] = 1;
06738
06739         /* Write data... */
06740         fprintf(out, "%.2f %g %g %g",
06741             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
06742         for (int iq = 0; iq < ctl->nq; iq++) {
06743             fprintf(out, " ");
06744             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
06745         }
06746         fprintf(out, "\n");
06747     }

```

```

06748
06749  /* Close file... */
06750  if (t == ctl->t_stop)
06751      fclose(out);
06752  }

```

Here is the call graph for this function:



## 5.20 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) 2013-2022 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  double clim_hno3(
00044      clim_t * clim,
00045      double t,
00046      double lat,
00047      double p) {
00048
00049      /* Get seconds since begin of year... */
00050      double sec = FMOD(t, 365.25 * 86400.);
00051      while (sec < 0)
00052          sec += 365.25 * 86400.;
00053
00054      /* Check pressure... */
00055      if (p < clim->hno3_p[0])
00056          p = clim->hno3_p[0];
00057      else if (p > clim->hno3_p[clim->hno3_np - 1])
00058          p = clim->hno3_p[clim->hno3_np - 1];
00059
00060      /* Check latitude... */
00061      if (lat < clim->hno3_lat[0])
00062          lat = clim->hno3_lat[0];
00063      else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])

```

```

00064     lat = clim->hno3_lat[clim->hno3_nlat - 1];
00065
00066     /* Get indices... */
00067     int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
00068     int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
00069     int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);
00070
00071     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00072     double aux00 = LIN(clim->hno3_p[ip],
00073                       clim->hno3[isec][ilat][ip],
00074                       clim->hno3_p[ip + 1],
00075                       clim->hno3[isec][ilat][ip + 1], p);
00076     double aux01 = LIN(clim->hno3_p[ip],
00077                       clim->hno3[isec][ilat + 1][ip],
00078                       clim->hno3_p[ip + 1],
00079                       clim->hno3[isec][ilat + 1][ip + 1], p);
00080     double aux10 = LIN(clim->hno3_p[ip],
00081                       clim->hno3[isec + 1][ilat][ip],
00082                       clim->hno3_p[ip + 1],
00083                       clim->hno3[isec + 1][ilat][ip + 1], p);
00084     double aux11 = LIN(clim->hno3_p[ip],
00085                       clim->hno3[isec + 1][ilat + 1][ip],
00086                       clim->hno3_p[ip + 1],
00087                       clim->hno3[isec + 1][ilat + 1][ip + 1], p);
00088     aux00 = LIN(clim->hno3_lat[ilat], aux00,
00089               clim->hno3_lat[ilat + 1], aux01, lat);
00090     aux11 = LIN(clim->hno3_lat[ilat], aux10,
00091               clim->hno3_lat[ilat + 1], aux11, lat);
00092     aux00 = LIN(clim->hno3_time[isec], aux00,
00093               clim->hno3_time[isec + 1], aux11, sec);
00094
00095     /* Convert from ppb to ppv... */
00096     return GSL_MAX(1e-9 * aux00, 0.0);
00097 }
00098
00099 /*****
00100
00101 void clim_hno3_init(
00102     clim_t * clim) {
00103
00104     /* Write info... */
00105     LOG(1, "Initialize HNO3 data...");
00106
00107     clim->hno3_ntime = 12;
00108     double hno3_time[12] = {
00109         1209600.00, 3888000.00, 6393600.00,
00110         9072000.00, 11664000.00, 14342400.00,
00111         16934400.00, 19612800.00, 22291200.00,
00112         24883200.00, 27561600.00, 30153600.00
00113     };
00114     memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));
00115
00116     clim->hno3_nlat = 18;
00117     double hno3_lat[18] = {
00118         -85, -75, -65, -55, -45, -35, -25, -15, -5,
00119         5, 15, 25, 35, 45, 55, 65, 75, 85
00120     };
00121     memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));
00122
00123     clim->hno3_np = 10;
00124     double hno3_p[10] = {
00125         4.64159, 6.81292, 10, 14.678, 21.5443,
00126         31.6228, 46.4159, 68.1292, 100, 146.78
00127     };
00128     memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));
00129
00130     double hno3[12][18][10] = {
00131         {0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00132         {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00133         {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00134         {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00135         {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00136         {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00137         {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00138         {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00139         {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00140         {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00141         {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00142         {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00143         {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00144         {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00145         {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00146         {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00147         {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00148         {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19},
00149         {0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00150         {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},

```

```
00151 {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00152 {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00153 {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00154 {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00155 {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00156 {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00157 {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00158 {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00159 {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00160 {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00161 {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00162 {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00163 {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00164 {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00165 {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00166 {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17},
00167 {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00168 {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00169 {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00170 {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00171 {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00172 {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00173 {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00174 {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00175 {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00176 {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00177 {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00178 {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00179 {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00180 {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00181 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00182 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00183 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00184 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42},
00185 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00186 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00187 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00188 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00189 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00190 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00191 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00192 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00193 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00194 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00195 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00196 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00197 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00198 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00199 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00200 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00201 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00202 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62},
00203 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00204 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00205 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00206 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00207 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00208 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00209 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00210 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00211 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00212 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00213 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00214 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00215 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00216 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00217 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00218 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00219 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00220 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00221 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00222 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00223 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00224 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00225 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00226 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00227 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00228 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00229 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00230 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00231 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00232 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00233 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00234 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00235 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00236 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00237 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
```



```

00238     {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00239     {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}},
00240     {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78}},
00241     {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08}},
00242     {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}},
00243     {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38}},
00244     {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}},
00245     {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176}},
00246     {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705}},
00247     {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}},
00248     {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199}},
00249     {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}},
00250     {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259}},
00251     {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422}},
00252     {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913}},
00253     {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4}},
00254     {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56}},
00255     {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61}},
00256     {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00257     {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}},
00258     {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}},
00259     {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6}},
00260     {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}},
00261     {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}},
00262     {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672}},
00263     {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19}},
00264     {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181}},
00265     {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107}},
00266     {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185}},
00267     {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224}},
00268     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232}},
00269     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341}},
00270     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754}},
00271     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23}},
00272     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45}},
00273     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5}},
00274     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00275     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56}},
00276     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74}},
00277     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09}},
00278     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83}},
00279     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22}},
00280     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646}},
00281     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169}},
00282     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587}},
00283     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815}},
00284     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147}},
00285     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197}},
00286     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163}},
00287     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303}},
00288     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714}},
00289     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1}},
00290     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41}},
00291     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56}},
00292     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00293     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91}},
00294     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84}},
00295     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97}},
00296     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}},
00297     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11}},
00298     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616}},
00299     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21}},
00300     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968}},
00301     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105}},
00302     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146}},
00303     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178}},
00304     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14}},
00305     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353}},
00306     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802}},
00307     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}},
00308     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52}},
00309     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73}},
00310     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00311     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}},
00312     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}},
00313     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75}},
00314     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41}},
00315     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955}},
00316     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61}},
00317     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269}},
00318     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132}},
00319     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121}},
00320     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}},
00321     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146}},
00322     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}},
00323     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448}},
00324     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948}},

```

```

00325     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00326     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00327     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00328     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00329     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00330     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74}},
00331     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00332     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00333     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00334     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00335     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00336     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00337     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00338     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00339     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00340     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00341     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00342     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00343     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00344     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00345     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00346     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00347 };
00348 memcpy(clim->hno3, hno3, sizeof(clim->hno3));
00349
00350 /* Get range... */
00351 double hno3min = 1e99, hno3max = -1e99;
00352 for (int it = 0; it < clim->hno3_ntime; it++)
00353     for (int iz = 0; iz < clim->hno3_np; iz++)
00354         for (int iy = 0; iy < clim->hno3_nlat; iy++) {
00355             hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
00356             hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
00357         }
00358
00359 /* Write info... */
00360 LOG(2, "Number of time steps: %d", clim->hno3_ntime);
00361 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00362     clim->hno3_time[0], clim->hno3_time[1],
00363     clim->hno3_time[clim->hno3_ntime - 1]);
00364 LOG(2, "Number of pressure levels: %d", clim->hno3_np);
00365 LOG(2, "Altitude levels: %g, %g ... %g km",
00366     Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
00367     Z(clim->hno3_p[clim->hno3_np - 1]));
00368 LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
00369     clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
00370 LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
00371 LOG(2, "Latitudes: %g, %g ... %g deg",
00372     clim->hno3_lat[0], clim->hno3_lat[1],
00373     clim->hno3_lat[clim->hno3_nlat - 1]);
00374 LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
00375     1e-9 * hno3max);
00376 }
00377
00378 /*****
00379
00380 double clim_oh(
00381     clim_t * clim,
00382     double t,
00383     double lat,
00384     double p) {
00385
00386     /* Get seconds since begin of year... */
00387     double sec = FMOD(t, 365.25 * 86400.);
00388     while (sec < 0)
00389         sec += 365.25 * 86400.;
00390
00391     /* Check pressure... */
00392     if (p < clim->oh_p[clim->oh_np - 1])
00393         p = clim->oh_p[clim->oh_np - 1];
00394     else if (p > clim->oh_p[0])
00395         p = clim->oh_p[0];
00396
00397     /* Check latitude... */
00398     if (lat < clim->oh_lat[0])
00399         lat = clim->oh_lat[0];
00400     else if (lat > clim->oh_lat[clim->oh_nlat - 1])
00401         lat = clim->oh_lat[clim->oh_nlat - 1];
00402
00403     /* Get indices... */
00404     int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
00405     int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
00406     int ip = locate_irr(clim->oh_p, clim->oh_np, p);
00407
00408     /* Interpolate OH climatology... */
00409     double aux00 = LIN(clim->oh_p[ip],
00410         clim->oh[isec][ip][ilat],
00411         clim->oh_p[ip + 1],

```

```

00412         clim->oh[isec][ip + 1][ilat], p);
00413     double aux01 = LIN(clim->oh_p[ip],
00414         clim->oh[isec][ip][ilat + 1],
00415         clim->oh_p[ip + 1],
00416         clim->oh[isec][ip + 1][ilat + 1], p);
00417     double aux10 = LIN(clim->oh_p[ip],
00418         clim->oh[isec + 1][ip][ilat],
00419         clim->oh_p[ip + 1],
00420         clim->oh[isec + 1][ip + 1][ilat], p);
00421     double aux11 = LIN(clim->oh_p[ip],
00422         clim->oh[isec + 1][ip][ilat + 1],
00423         clim->oh_p[ip + 1],
00424         clim->oh[isec + 1][ip + 1][ilat + 1], p);
00425     aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
00426     aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
00427     aux00 =
00428         LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);
00429
00430     return GSL_MAX(aux00, 0.0);
00431 }
00432
00433 /*****
00434
00435 double clim_oh_diurnal(
00436     ctl_t * ctl,
00437     clim_t * clim,
00438     double t,
00439     double p,
00440     double lon,
00441     double lat) {
00442
00443     double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);
00444
00445     if (sza2 <= M_PI / 2. * 89. / 90.)
00446         return oh * exp(-ctl->oh_chem_beta / cos(sza2));
00447     else
00448         return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
00449 }
00450
00451 /*****
00452
00453 void clim_oh_init(
00454     ctl_t * ctl,
00455     clim_t * clim) {
00456
00457     int nt, ncid, varid;
00458
00459     double *help, ohmin = 1e99, ohmax = -1e99;
00460
00461     /* Write info... */
00462     LOG(1, "Read OH data: %s", ctl->clim_oh_filename);
00463
00464     /* Open netCDF file... */
00465     if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00466         WARN("OH climatology data are missing!");
00467         return;
00468     }
00469
00470     /* Read pressure data... */
00471     NC_INQ_DIM("press", &clim->oh_np, 2, CP);
00472     NC_GET_DOUBLE("press", clim->oh_p, 1);
00473
00474     /* Check ordering of pressure data... */
00475     if (clim->oh_p[0] < clim->oh_p[1])
00476         ERRMSG("Pressure data are not descending!");
00477
00478     /* Read latitudes... */
00479     NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
00480     NC_GET_DOUBLE("lat", clim->oh_lat, 1);
00481
00482     /* Check ordering of latitudes... */
00483     if (clim->oh_lat[0] > clim->oh_lat[1])
00484         ERRMSG("Latitude data are not ascending!");
00485
00486     /* Set time data for monthly means... */
00487     clim->oh_ntime = 12;
00488     clim->oh_time[0] = 1209600.00;
00489     clim->oh_time[1] = 3888000.00;
00490     clim->oh_time[2] = 6393600.00;
00491     clim->oh_time[3] = 9072000.00;
00492     clim->oh_time[4] = 11664000.00;
00493     clim->oh_time[5] = 14342400.00;
00494     clim->oh_time[6] = 16934400.00;
00495     clim->oh_time[7] = 19612800.00;
00496     clim->oh_time[8] = 22291200.00;
00497     clim->oh_time[9] = 24883200.00;
00498     clim->oh_time[10] = 27561600.00;

```

```

00499 clim->oh_time[11] = 30153600.00;
00500
00501 /* Check number of timesteps... */
00502 NC_INQ_DIM("time", &nt, 12, 12);
00503
00504 /* Read OH data... */
00505 ALLOC(help, double,
00506        clim->oh_nlat * clim->oh_np * clim->oh_ntime);
00507 NC_GET_DOUBLE("OH", help, 1);
00508 for (int it = 0; it < clim->oh_ntime; it++)
00509     for (int iz = 0; iz < clim->oh_np; iz++)
00510         for (int iy = 0; iy < clim->oh_nlat; iy++) {
00511             clim->oh[it][iz][iy] =
00512                 help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
00513                 / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
00514                                     clim->oh_lat[iy]);
00515             ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
00516             ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
00517         }
00518 free(help);
00519
00520 /* Close netCDF file... */
00521 NC(nc_close(ncid));
00522
00523 /* Write info... */
00524 LOG(2, "Number of time steps: %d", clim->oh_ntime);
00525 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00526        clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
00527 LOG(2, "Number of pressure levels: %d", clim->oh_np);
00528 LOG(2, "Altitude levels: %g, %g ... %g km",
00529        Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
00530 LOG(2, "Pressure levels: %g, %g ... %g hPa",
00531        clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
00532 LOG(2, "Number of latitudes: %d", clim->oh_nlat);
00533 LOG(2, "Latitudes: %g, %g ... %g deg",
00534        clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
00535 LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
00536 }
00537
00538 /*****
00539
00540 double clim_oh_init_help(
00541     double beta,
00542     double time,
00543     double lat) {
00544
00545     double aux, lon, sum = 0;
00546
00547     int n = 0;
00548
00549     /* Integrate day/night correction factor over longitude... */
00550     for (lon = -180; lon < 180; lon += 1) {
00551         aux = sza(time, lon, lat);
00552         if (aux <= M_PI / 2. * 85. / 90.)
00553             sum += exp(-beta / cos(aux));
00554         else
00555             sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
00556         n++;
00557     }
00558     return sum / (double) n;
00559 }
00560
00561 /*****
00562
00563 double clim_h2o2(
00564     clim_t * clim,
00565     double t,
00566     double lat,
00567     double p) {
00568
00569     /* Get seconds since begin of year... */
00570     double sec = FMOD(t, 365.25 * 86400.);
00571     while (sec < 0)
00572         sec += 365.25 * 86400.;
00573
00574     /* Check pressure... */
00575     if (p < clim->h2o2_p[clim->h2o2_np - 1])
00576         p = clim->h2o2_p[clim->h2o2_np - 1];
00577     else if (p > clim->h2o2_p[0])
00578         p = clim->h2o2_p[0];
00579
00580     /* Check latitude... */
00581     if (lat < clim->h2o2_lat[0])
00582         lat = clim->h2o2_lat[0];
00583     else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
00584         lat = clim->h2o2_lat[clim->h2o2_nlat - 1];
00585

```

```

00586  /* Get indices... */
00587  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
00588  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
00589  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);
00590
00591  /* Interpolate H2O2 climatology... */
00592  double aux00 = LIN(clim->h2o2_p[ip],
00593                    clim->h2o2[isec][ip][ilat],
00594                    clim->h2o2_p[ip + 1],
00595                    clim->h2o2[isec][ip + 1][ilat], p);
00596  double aux01 = LIN(clim->h2o2_p[ip],
00597                    clim->h2o2[isec][ip][ilat + 1],
00598                    clim->h2o2_p[ip + 1],
00599                    clim->h2o2[isec][ip + 1][ilat + 1], p);
00600  double aux10 = LIN(clim->h2o2_p[ip],
00601                    clim->h2o2[isec + 1][ip][ilat],
00602                    clim->h2o2_p[ip + 1],
00603                    clim->h2o2[isec + 1][ip + 1][ilat], p);
00604  double aux11 = LIN(clim->h2o2_p[ip],
00605                    clim->h2o2[isec + 1][ip][ilat + 1],
00606                    clim->h2o2_p[ip + 1],
00607                    clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
00608  aux00 =
00609      LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
00610  aux11 =
00611      LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
00612  aux00 =
00613      LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);
00614
00615  return GSL_MAX(aux00, 0.0);
00616 }
00617
00618 /*****
00619
00620 void clim_h2o2_init(
00621     ctl_t * ctl,
00622     clim_t * clim) {
00623
00624     int ncid, varid, it, iy, iz, nt;
00625
00626     double *help, h2o2min = 1e99, h2o2max = -1e99;
00627
00628     /* Write info... */
00629     LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);
00630
00631     /* Open netCDF file... */
00632     if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00633         WARN("H2O2 climatology data are missing!");
00634         return;
00635     }
00636
00637     /* Read pressure data... */
00638     NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
00639     NC_GET_DOUBLE("press", clim->h2o2_p, 1);
00640
00641     /* Check ordering of pressure data... */
00642     if (clim->h2o2_p[0] < clim->h2o2_p[1])
00643         ERRMSG("Pressure data are not descending!");
00644
00645     /* Read latitudes... */
00646     NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
00647     NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);
00648
00649     /* Check ordering of latitude data... */
00650     if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
00651         ERRMSG("Latitude data are not ascending!");
00652
00653     /* Set time data (for monthly means)... */
00654     clim->h2o2_ntime = 12;
00655     clim->h2o2_time[0] = 1209600.00;
00656     clim->h2o2_time[1] = 3888000.00;
00657     clim->h2o2_time[2] = 6393600.00;
00658     clim->h2o2_time[3] = 9072000.00;
00659     clim->h2o2_time[4] = 11664000.00;
00660     clim->h2o2_time[5] = 14342400.00;
00661     clim->h2o2_time[6] = 16934400.00;
00662     clim->h2o2_time[7] = 19612800.00;
00663     clim->h2o2_time[8] = 22291200.00;
00664     clim->h2o2_time[9] = 24883200.00;
00665     clim->h2o2_time[10] = 27561600.00;
00666     clim->h2o2_time[11] = 30153600.00;
00667
00668     /* Check number of timesteps... */
00669     NC_INQ_DIM("time", &nt, 12, 12);
00670
00671     /* Read data... */
00672     ALLOC(help, double,

```

```

00673     clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
00674 NC_GET_DOUBLE("h2o2", help, 1);
00675 for (it = 0; it < clim->h2o2_ntime; it++)
00676     for (iz = 0; iz < clim->h2o2_np; iz++)
00677         for (iy = 0; iy < clim->h2o2_nlat; iy++) {
00678             clim->h2o2[it][iz][iy] =
00679                 help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
00680             h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
00681             h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
00682         }
00683 free(help);
00684
00685 /* Close netCDF file... */
00686 NC(nc_close(ncid));
00687
00688 /* Write info... */
00689 LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
00690 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00691     clim->h2o2_time[0], clim->h2o2_time[1],
00692     clim->h2o2_time[clim->h2o2_ntime - 1]);
00693 LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
00694 LOG(2, "Altitude levels: %g, %g ... %g km",
00695     Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
00696     Z(clim->h2o2_p[clim->h2o2_np - 1]));
00697 LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
00698     clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
00699 LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
00700 LOG(2, "Latitudes: %g, %g ... %g deg",
00701     clim->h2o2_lat[0], clim->h2o2_lat[1],
00702     clim->h2o2_lat[clim->h2o2_nlat - 1]);
00703 LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
00704 }
00705
00706 /*****
00707
00708 double clim_tropo(
00709     clim_t * clim,
00710     double t,
00711     double lat) {
00712
00713     /* Get seconds since begin of year... */
00714     double sec = FMOD(t, 365.25 * 86400.);
00715     while (sec < 0)
00716         sec += 365.25 * 86400.;
00717
00718     /* Get indices... */
00719     int isec = locate_irr(clim->tropo_time, clim->tropo_ntime, sec);
00720     int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);
00721
00722     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
00723     double p0 = LIN(clim->tropo_lat[ilat],
00724         clim->tropo[isec][ilat],
00725         clim->tropo_lat[ilat + 1],
00726         clim->tropo[isec][ilat + 1], lat);
00727     double p1 = LIN(clim->tropo_lat[ilat],
00728         clim->tropo[isec + 1][ilat],
00729         clim->tropo_lat[ilat + 1],
00730         clim->tropo[isec + 1][ilat + 1], lat);
00731     return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
00732 }
00733
00734 /*****
00735
00736 void clim_tropo_init(
00737     clim_t * clim) {
00738
00739     /* Write info... */
00740     LOG(1, "Initialize tropopause data...");
00741
00742     clim->tropo_ntime = 12;
00743     double tropo_time[12] = {
00744         1209600.00, 3888000.00, 6393600.00,
00745         9072000.00, 11664000.00, 14342400.00,
00746         16934400.00, 19612800.00, 22291200.00,
00747         24883200.00, 27561600.00, 30153600.00
00748     };
00749     memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));
00750
00751     clim->tropo_nlat = 73;
00752     double tropo_lat[73] = {
00753         -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00754         -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00755         -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00756         -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00757         15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00758         45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00759         75, 77.5, 80, 82.5, 85, 87.5, 90

```

```
00760     };
00761     memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));
00762
00763     double tropo[12][73] = {
00764         {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00765          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00766          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00767          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00768          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00769          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00770          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00771          275.3, 275.6, 275.4, 274.1, 273.5},
00772         {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00773          300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00774          150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00775          98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00776          98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00777          220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00778          284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00779          287.5, 286.2, 285.8},
00780         {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00781          297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00782          161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00783          100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00784          99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00785          186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00786          279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00787          304.3, 304.9, 306, 306.6, 306.2, 306},
00788         {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00789          290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00790          195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00791          102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00792          99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00793          148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00794          263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00795          315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00796         {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00797          260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00798          205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00799          101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00800          102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00801          165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00802          273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00803          325.3, 325.8, 325.8},
00804         {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00805          222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00806          228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00807          105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00808          106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00809          127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00810          251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00811          308.5, 312.2, 313.1, 313.3},
00812         {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00813          187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00814          235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00815          110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00816          111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00817          117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00818          224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00819          275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00820         {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00821          185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00822          233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00823          110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00824          112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00825          120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00826          230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00827          278.2, 282.6, 287.4, 290.9, 292.5, 293},
00828         {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00829          183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00830          243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00831          114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00832          110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00833          114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00834          203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00835          276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00836         {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00837          215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00838          237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00839          111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00840          106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00841          112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00842          206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00843          279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00844          305.1},
00845         {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00846          253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
```

```

00847     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00848     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00849     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00850     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00851     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00852     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00853     {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00854     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00855     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00856     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00857     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00858     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00859     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00860     281.7, 281.1, 281.2}
00861 };
00862 memcpy(clim->tropo, tropo, sizeof(clim->tropo));
00863
00864 /* Get range... */
00865 double tropomin = 1e99, tropomax = -1e99;
00866 for (int it = 0; it < clim->tropo_ntime; it++)
00867     for (int iy = 0; iy < clim->tropo_nlat; iy++) {
00868         tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
00869         tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
00870     }
00871
00872 /* Write info... */
00873 LOG(2, "Number of time steps: %d", clim->tropo_ntime);
00874 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00875     clim->tropo_time[0], clim->tropo_time[1],
00876     clim->tropo_time[clim->tropo_ntime - 1]);
00877 LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
00878 LOG(2, "Latitudes: %g, %g ... %g deg",
00879     clim->tropo_lat[0], clim->tropo_lat[1],
00880     clim->tropo_lat[clim->tropo_nlat - 1]);
00881 LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
00882     Z(tropomin));
00883 LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
00884 }
00885
00886 /*****
00887 void compress_pack(
00888     char *varname,
00889     float *array,
00890     size_t nxy,
00891     size_t nz,
00892     int decompress,
00893     FILE * inout) {
00894
00895     double min[EP], max[EP], off[EP], scl[EP];
00896
00897     unsigned short *sarray;
00898
00899     /* Allocate... */
00900     ALLOC(sarray, unsigned short,
00901         nxy * nz);
00902
00903     /* Read compressed stream and decompress array... */
00904     if (decompress) {
00905
00906         /* Write info... */
00907         LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
00908             varname, 100. * sizeof(unsigned short) / sizeof(float));
00909
00910         /* Read data... */
00911         FREAD(&scl, double,
00912             nz,
00913             inout);
00914         FREAD(&off, double,
00915             nz,
00916             inout);
00917         FREAD(sarray, unsigned short,
00918             nxy * nz,
00919             inout);
00920
00921         /* Convert to float... */
00922         #pragma omp parallel for default(shared)
00923         for (size_t ixy = 0; ixy < nxy; ixy++)
00924             for (size_t iz = 0; iz < nz; iz++)
00925                 array[ixy * nz + iz]
00926                     = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
00927     }
00928
00929     /* Compress array and output compressed stream... */
00930     else {
00931
00932         /* Write info... */

```



```

00934     LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
00935         varname, 100. * sizeof(unsigned short) / sizeof(float));
00936
00937     /* Get range... */
00938     for (size_t iz = 0; iz < nz; iz++) {
00939         min[iz] = array[iz];
00940         max[iz] = array[iz];
00941     }
00942     for (size_t ixy = 1; ixy < nxy; ixy++)
00943         for (size_t iz = 0; iz < nz; iz++) {
00944             if (array[ixy * nz + iz] < min[iz])
00945                 min[iz] = array[ixy * nz + iz];
00946             if (array[ixy * nz + iz] > max[iz])
00947                 max[iz] = array[ixy * nz + iz];
00948         }
00949
00950     /* Get offset and scaling factor... */
00951     for (size_t iz = 0; iz < nz; iz++) {
00952         scl[iz] = (max[iz] - min[iz]) / 65533.;
00953         off[iz] = min[iz];
00954     }
00955
00956     /* Convert to short... */
00957     #pragma omp parallel for default(shared)
00958     for (size_t ixy = 0; ixy < nxy; ixy++)
00959         for (size_t iz = 0; iz < nz; iz++)
00960             if (scl[iz] != 0)
00961                 sarray[ixy * nz + iz] = (unsigned short)
00962                     ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
00963             else
00964                 sarray[ixy * nz + iz] = 0;
00965
00966     /* Write data... */
00967     FWRITE(&scl, double,
00968         nz,
00969         inout);
00970     FWRITE(&off, double,
00971         nz,
00972         inout);
00973     FWRITE(sarray, unsigned short,
00974         nxy * nz,
00975         inout);
00976 }
00977
00978 /* Free... */
00979 free(sarray);
00980 }
00981
00982 /*****
00983
00984 #ifdef ZFP
00985 void compress_zfp(
00986     char *varname,
00987     float *array,
00988     int nx,
00989     int ny,
00990     int nz,
00991     int precision,
00992     double tolerance,
00993     int decompress,
00994     FILE * inout) {
00995
00996     zfp_type type;                /* array scalar type */
00997     zfp_field *field;             /* array meta data */
00998     zfp_stream *zfp;             /* compressed stream */
00999     void *buffer;                /* storage for compressed stream */
01000     size_t bufsize;              /* byte size of compressed buffer */
01001     bitstream *stream;           /* bit stream to write to or read from */
01002     size_t zfpsize;              /* byte size of compressed stream */
01003
01004     /* Allocate meta data for the 3D array a[nz][ny][nx]... */
01005     type = zfp_type_float;
01006     field = zfp_field_3d(array, type, (uint) nx, (uint) ny, (uint) nz);
01007
01008     /* Allocate meta data for a compressed stream... */
01009     zfp = zfp_stream_open(NULL);
01010
01011     /* Set compression mode... */
01012     int actual_prec = 0;
01013     double actual_tol = 0;
01014     if (precision > 0)
01015         actual_prec = (int) zfp_stream_set_precision(zfp, (uint) precision);
01016     else if (tolerance > 0)
01017         actual_tol = zfp_stream_set_accuracy(zfp, tolerance);
01018     else
01019         ERRMSG("Set precision or tolerance!");
01020

```

```

01021  /* Allocate buffer for compressed data... */
01022  bufsize = zfp_stream_maximum_size(zfp, field);
01023  buffer = malloc(bufsize);
01024
01025  /* Associate bit stream with allocated buffer... */
01026  stream = stream_open(buffer, bufsize);
01027  zfp_stream_set_bit_stream(zfp, stream);
01028  zfp_stream_rewind(zfp);
01029
01030  /* Read compressed stream and decompress array... */
01031  if (decompress) {
01032      FREAD(&zfp_size, size_t,
01033           1,
01034           inout);
01035      if (fread(buffer, 1, zfp_size, inout) != zfp_size)
01036          ERRMSG("Error while reading zfp data!");
01037      if (!zfp_decompress(zfp, field)) {
01038          ERRMSG("Decompression failed!");
01039      }
01040      LOG(2, "Read 3-D variable: %s "
01041           "(zfp, PREC= %d, TOL= %g, RATIO= %g %%)",
01042           varname, actual_prec, actual_tol,
01043           (100. * (double) zfp_size) / (double) (nx * ny * nz));
01044  }
01045
01046  /* Compress array and output compressed stream... */
01047  else {
01048      zfp_size = zfp_compress(zfp, field);
01049      if (!zfp_size) {
01050          ERRMSG("Compression failed!");
01051      } else {
01052          FWRITE(&zfp_size, size_t,
01053               1,
01054               inout);
01055          if (fwrite(buffer, 1, zfp_size, inout) != zfp_size)
01056              ERRMSG("Error while writing zfp data!");
01057      }
01058      LOG(2, "Write 3-D variable: %s "
01059           "(zfp, PREC= %d, TOL= %g, RATIO= %g %%)",
01060           varname, actual_prec, actual_tol,
01061           (100. * (double) zfp_size) / (double) (nx * ny * nz));
01062  }
01063
01064  /* Free... */
01065  zfp_field_free(field);
01066  zfp_stream_close(zfp);
01067  stream_close(stream);
01068  free(buffer);
01069 }
01070 #endif
01071
01072 /*****
01073
01074 #ifdef ZSTD
01075 void compress_zstd(
01076     char *varname,
01077     float *array,
01078     size_t n,
01079     int decompress,
01080     FILE * inout) {
01081
01082     /* Get buffer sizes... */
01083     size_t uncomprLen = n * sizeof(float);
01084     size_t comprLen = ZSTD_compressBound(uncomprLen);
01085     size_t compsize;
01086
01087     /* Allocate... */
01088     char *compr = (char *) calloc((uint) comprLen, 1);
01089     char *uncompr = (char *) array;
01090
01091     /* Read compressed stream and decompress array... */
01092     if (decompress) {
01093         FREAD(&comprLen, size_t,
01094              1,
01095              inout);
01096         if (fread(compr, 1, comprLen, inout) != comprLen)
01097             ERRMSG("Error while reading zstd data!");
01098         compsize = ZSTD_decompress(uncompr, uncomprLen, compr, comprLen);
01099         if (ZSTD_isError(compsize)) {
01100             ERRMSG("Decompression failed!");
01101         }
01102         LOG(2, "Read 3-D variable: %s (zstd, RATIO= %g %%)",
01103              varname, (100. * (double) comprLen) / (double) uncomprLen);
01104     }
01105
01106     /* Compress array and output compressed stream... */
01107     else {

```

```

01108     compsize = ZSTD_compress(compr, comprLen, uncompr, uncomprLen, 0);
01109     if (ZSTD_isError(compsize)) {
01110         ERRMSG("Compression failed!");
01111     } else {
01112         FWRITE(&compsize, size_t,
01113             1,
01114             inout);
01115         if (fwrite(compr, 1, compsize, inout) != compsize)
01116             ERRMSG("Error while writing zstd data!");
01117     }
01118     LOG(2, "Write 3-D variable: %s (zstd, RATIO= %g %%)",
01119         varname, (100. * (double) compsize) / (double) uncomprLen);
01120 }
01121
01122 /* Free... */
01123 free(compr);
01124 }
01125 #endif
01126
01127 /*****
01128
01129 void day2doy(
01130     int year,
01131     int mon,
01132     int day,
01133     int *doy) {
01134
01135     const int
01136         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01137         d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01138
01139     /* Get day of year... */
01140     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01141         *doy = d0l[mon - 1] + day - 1;
01142     else
01143         *doy = d0[mon - 1] + day - 1;
01144 }
01145
01146 /*****
01147
01148 void doy2day(
01149     int year,
01150     int doy,
01151     int *mon,
01152     int *day) {
01153
01154     const int
01155         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01156         d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01157
01158     int i;
01159
01160     /* Get month and day... */
01161     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01162         for (i = 11; i > 0; i--)
01163             if (d0l[i] <= doy)
01164                 break;
01165         *mon = i + 1;
01166         *day = doy - d0l[i] + 1;
01167     } else {
01168         for (i = 11; i > 0; i--)
01169             if (d0[i] <= doy)
01170                 break;
01171         *mon = i + 1;
01172         *day = doy - d0[i] + 1;
01173     }
01174 }
01175
01176 /*****
01177
01178 void geo2cart(
01179     double z,
01180     double lon,
01181     double lat,
01182     double *x) {
01183
01184     double radius = z + RE;
01185     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01186     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01187     x[2] = radius * sin(lat / 180. * M_PI);
01188 }
01189
01190 /*****
01191
01192 void get_met(
01193     ctl_t * ctl,
01194     clim_t * clim,

```

```

01195     double t,
01196     met_t ** met0,
01197     met_t ** met1) {
01198
01199     static int init;
01200
01201     met_t *mets;
01202
01203     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01204
01205     /* Set timer... */
01206     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01207
01208     /* Init... */
01209     if (t == ctl->t_start || !init) {
01210         init = 1;
01211
01212         /* Read meteo data... */
01213         get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
01214                     ctl->metbase, ctl->dt_met, filename);
01215         if (!read_met(filename, ctl, clim, *met0))
01216             ERRMSG("Cannot open file!");
01217
01218         get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
01219                     ctl->metbase, ctl->dt_met, filename);
01220         if (!read_met(filename, ctl, clim, *met1))
01221             ERRMSG("Cannot open file!");
01222
01223         /* Update GPU... */
01224 #ifdef _OPENACC
01225         met_t *met0up = *met0;
01226         met_t *met1up = *met1;
01227 #ifdef ASYNCIO
01228 #pragma acc update device(met0up[:1],met1up[:1]) async(5)
01229 #else
01230 #pragma acc update device(met0up[:1],met1up[:1])
01231 #endif
01232 #endif
01233
01234         /* Caching... */
01235         if (ctl->met_cache && t != ctl->t_stop) {
01236             get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
01237                         ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
01238             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01239             LOG(1, "Caching: %s", cachefile);
01240             if (system(cmd) != 0)
01241                 WARN("Caching command failed!");
01242         }
01243     }
01244
01245     /* Read new data for forward trajectories... */
01246     if (t > (*met1)->time) {
01247
01248         /* Pointer swap... */
01249         mets = *met1;
01250         *met1 = *met0;
01251         *met0 = mets;
01252
01253         /* Read new meteo data... */
01254         get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
01255         if (!read_met(filename, ctl, clim, *met1))
01256             ERRMSG("Cannot open file!");
01257         /* Update GPU... */
01258 #ifdef _OPENACC
01259         met_t *met1up = *met1;
01260 #ifdef ASYNCIO
01261 #pragma acc update device(met1up[:1]) async(5)
01262 #else
01263 #pragma acc update device(met1up[:1])
01264 #endif
01265 #endif
01266
01267         /* Caching... */
01268         if (ctl->met_cache && t != ctl->t_stop) {
01269             get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
01270                         cachefile);
01271             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01272             LOG(1, "Caching: %s", cachefile);
01273             if (system(cmd) != 0)
01274                 WARN("Caching command failed!");
01275         }
01276
01277         /* Read new data for backward trajectories... */
01278         if (t < (*met0)->time) {
01279
01280             /* Pointer swap... */
01281             mets = *met1;
01282             *met1 = *met0;

```

```

01282     *met0 = mets;
01283
01284     /* Read new meteo data... */
01285     get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
01286     if (!read_met(filename, ctl, clim, *met0))
01287         ERRMSG("Cannot open file!");
01288
01289     /* Update GPU... */
01290 #ifdef _OPENACC
01291     met_t *met0up = *met0;
01292 #ifdef ASYNCIO
01293     #pragma acc update device(met0up[:1]) async(5)
01294 #else
01295     #pragma acc update device(met0up[:1])
01296 #endif
01297 #endif
01298
01299     /* Caching... */
01300     if (ctl->met_cache && t != ctl->t_stop) {
01301         get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
01302             cachefile);
01303         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01304         LOG(1, "Caching: %s", cachefile);
01305         if (system(cmd) != 0)
01306             WARN("Caching command failed!");
01307     }
01308 }
01309
01310 /* Check that grids are consistent... */
01311 if ((*met0)->nx != 0 && (*met1)->nx != 0) {
01312     if ((*met0)->nx != (*met1)->nx
01313         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01314         ERRMSG("Meteo grid dimensions do not match!");
01315     for (int ix = 0; ix < (*met0)->nx; ix++)
01316         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01317             ERRMSG("Meteo grid longitudes do not match!");
01318     for (int iy = 0; iy < (*met0)->ny; iy++)
01319         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01320             ERRMSG("Meteo grid latitudes do not match!");
01321     for (int ip = 0; ip < (*met0)->np; ip++)
01322         if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01323             ERRMSG("Meteo grid pressure levels do not match!");
01324 }
01325
01326 /*****
01327
01328 void get_met_help(
01329     ctl_t * ctl,
01330     double t,
01331     int direct,
01332     char *metbase,
01333     double dt_met,
01334     char *filename) {
01335
01336     char repl[LEN];
01337
01338     double t6, r;
01339
01340     int year, mon, day, hour, min, sec;
01341
01342     /* Round time to fixed intervals... */
01343     if (direct == -1)
01344         t6 = floor(t / dt_met) * dt_met;
01345     else
01346         t6 = ceil(t / dt_met) * dt_met;
01347
01348     /* Decode time... */
01349     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01350
01351     /* Set filename of MPTRAC meteo files... */
01352     if (ctl->clams_met_data == 0) {
01353         if (ctl->met_type == 0)
01354             sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01355         else if (ctl->met_type == 1)
01356             sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
01357         else if (ctl->met_type == 2)
01358             sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
01359         else if (ctl->met_type == 3)
01360             sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
01361         else if (ctl->met_type == 4)
01362             sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", metbase);
01363         sprintf(repl, "%d", year);
01364         get_met_replace(filename, "YYYY", repl);
01365         sprintf(repl, "%02d", mon);
01366         get_met_replace(filename, "MM", repl);
01367         sprintf(repl, "%02d", day);
01368         get_met_replace(filename, "DD", repl);

```

```

01369     sprintf(repl, "%02d", hour);
01370     get_met_replace(filename, "HH", repl);
01371 }
01372
01373 /* Set filename of CLaMS meteo files... */
01374 else {
01375     sprintf(filename, "%s_YYMMDDHH.nc", metbase);
01376     sprintf(repl, "%d", year);
01377     get_met_replace(filename, "YYYY", repl);
01378     sprintf(repl, "%d", year % 100);
01379     get_met_replace(filename, "YY", repl);
01380     sprintf(repl, "%02d", mon);
01381     get_met_replace(filename, "MM", repl);
01382     sprintf(repl, "%02d", day);
01383     get_met_replace(filename, "DD", repl);
01384     sprintf(repl, "%02d", hour);
01385     get_met_replace(filename, "HH", repl);
01386 }
01387 }
01388
01389 /*****
01390
01391 void get_met_replace(
01392     char *orig,
01393     char *search,
01394     char *repl) {
01395
01396     char buffer[LEN];
01397
01398     /* Iterate... */
01399     for (int i = 0; i < 3; i++) {
01400
01401         /* Replace sub-string... */
01402         char *ch;
01403         if (!(ch = strstr(orig, search)))
01404             return;
01405         strncpy(buffer, orig, (size_t) (ch - orig));
01406         buffer[ch - orig] = 0;
01407         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01408         orig[0] = 0;
01409         strcpy(orig, buffer);
01410     }
01411 }
01412
01413 /*****
01414
01415 void intpol_met_space_3d(
01416     met_t * met,
01417     float array[EX][EY][EP],
01418     double p,
01419     double lon,
01420     double lat,
01421     double *var,
01422     int *ci,
01423     double *cw,
01424     int init) {
01425
01426     /* Initialize interpolation... */
01427     if (init) {
01428
01429         /* Check longitude... */
01430         if (met->lon[met->nx - 1] > 180 && lon < 0)
01431             lon += 360;
01432
01433         /* Get interpolation indices... */
01434         ci[0] = locate_irr(met->p, met->np, p);
01435         ci[1] = locate_reg(met->lon, met->nx, lon);
01436         ci[2] = locate_reg(met->lat, met->ny, lat);
01437
01438         /* Get interpolation weights... */
01439         cw[0] = (met->p[ci[0] + 1] - p)
01440             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01441         cw[1] = (met->lon[ci[1] + 1] - lon)
01442             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01443         cw[2] = (met->lat[ci[2] + 1] - lat)
01444             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01445     }
01446
01447     /* Interpolate vertically... */
01448     double aux00 =
01449         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01450         + array[ci[1]][ci[2]][ci[0] + 1];
01451     double aux01 =
01452         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01453             array[ci[1]][ci[2] + 1][ci[0] + 1])
01454         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01455     double aux10 =

```

```

01456     cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01457             array[ci[1] + 1][ci[2]][ci[0] + 1])
01458     + array[ci[1] + 1][ci[2]][ci[0] + 1];
01459 double aux11 =
01460     cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01461             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01462     + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01463
01464 /* Interpolate horizontally... */
01465 aux00 = cw[2] * (aux00 - aux01) + aux01;
01466 aux11 = cw[2] * (aux10 - aux11) + aux11;
01467 *var = cw[1] * (aux00 - aux11) + aux11;
01468 }
01469
01470 /*****
01471
01472 void intpol_met_space_2d(
01473     met_t * met,
01474     float array[EX][EY],
01475     double lon,
01476     double lat,
01477     double *var,
01478     int *ci,
01479     double *cw,
01480     int init) {
01481
01482     /* Initialize interpolation... */
01483     if (init) {
01484
01485         /* Check longitude... */
01486         if (met->lon[met->nx - 1] > 180 && lon < 0)
01487             lon += 360;
01488
01489         /* Get interpolation indices... */
01490         ci[1] = locate_reg(met->lon, met->nx, lon);
01491         ci[2] = locate_reg(met->lat, met->ny, lat);
01492
01493         /* Get interpolation weights... */
01494         cw[1] = (met->lon[ci[1] + 1] - lon)
01495             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01496         cw[2] = (met->lat[ci[2] + 1] - lat)
01497             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01498     }
01499
01500     /* Set variables... */
01501     double aux00 = array[ci[1]][ci[2]];
01502     double aux01 = array[ci[1]][ci[2] + 1];
01503     double aux10 = array[ci[1] + 1][ci[2]];
01504     double aux11 = array[ci[1] + 1][ci[2] + 1];
01505
01506     /* Interpolate horizontally... */
01507     if (isfinite(aux00) && isfinite(aux01)
01508         && isfinite(aux10) && isfinite(aux11)) {
01509         aux00 = cw[2] * (aux00 - aux01) + aux01;
01510         aux11 = cw[2] * (aux10 - aux11) + aux11;
01511         *var = cw[1] * (aux00 - aux11) + aux11;
01512     } else {
01513         if (cw[2] < 0.5) {
01514             if (cw[1] < 0.5)
01515                 *var = aux11;
01516             else
01517                 *var = aux01;
01518         } else {
01519             if (cw[1] < 0.5)
01520                 *var = aux10;
01521             else
01522                 *var = aux00;
01523         }
01524     }
01525 }
01526
01527 /*****
01528
01529 #ifdef UVW
01530 void intpol_met_space_uvw(
01531     met_t * met,
01532     double p,
01533     double lon,
01534     double lat,
01535     double *u,
01536     double *v,
01537     double *w,
01538     int *ci,
01539     double *cw,
01540     int init) {
01541
01542     /* Initialize interpolation... */

```

```

01543 if (init) {
01544
01545     /* Check longitude... */
01546     if (met->lon[met->nx - 1] > 180 && lon < 0)
01547         lon += 360;
01548
01549     /* Get interpolation indices... */
01550     ci[0] = locate_irr(met->p, met->np, p);
01551     ci[1] = locate_reg(met->lon, met->nx, lon);
01552     ci[2] = locate_reg(met->lat, met->ny, lat);
01553
01554     /* Get interpolation weights... */
01555     cw[0] = (met->p[ci[0] + 1] - p)
01556         / (met->p[ci[0] + 1] - met->p[ci[0]]);
01557     cw[1] = (met->lon[ci[1] + 1] - lon)
01558         / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01559     cw[2] = (met->lat[ci[2] + 1] - lat)
01560         / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01561 }
01562
01563 /* Interpolate vertically... */
01564 double u00 =
01565     cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][0] -
01566         met->uvw[ci[1]][ci[2]][ci[0] + 1][0])
01567     + met->uvw[ci[1]][ci[2]][ci[0] + 1][0];
01568 double u01 =
01569     cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][0] -
01570         met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][0])
01571     + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][0];
01572 double u10 =
01573     cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][0] -
01574         met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][0])
01575     + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][0];
01576 double u11 =
01577     cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][0] -
01578         met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][0])
01579     + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][0];
01580
01581 double v00 =
01582     cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][1] -
01583         met->uvw[ci[1]][ci[2]][ci[0] + 1][1])
01584     + met->uvw[ci[1]][ci[2]][ci[0] + 1][1];
01585 double v01 =
01586     cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][1] -
01587         met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][1])
01588     + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][1];
01589 double v10 =
01590     cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][1] -
01591         met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][1])
01592     + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][1];
01593 double v11 =
01594     cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][1] -
01595         met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][1])
01596     + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][1];
01597
01598 double w00 =
01599     cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][2] -
01600         met->uvw[ci[1]][ci[2]][ci[0] + 1][2])
01601     + met->uvw[ci[1]][ci[2]][ci[0] + 1][2];
01602 double w01 =
01603     cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][2] -
01604         met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][2])
01605     + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][2];
01606 double w10 =
01607     cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][2] -
01608         met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][2])
01609     + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][2];
01610 double w11 =
01611     cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][2] -
01612         met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][2])
01613     + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][2];
01614
01615 /* Interpolate horizontally... */
01616 u00 = cw[2] * (u00 - u01) + u01;
01617 u11 = cw[2] * (u10 - u11) + u11;
01618 *u = cw[1] * (u00 - u11) + u11;
01619
01620 v00 = cw[2] * (v00 - v01) + v01;
01621 v11 = cw[2] * (v10 - v11) + v11;
01622 *v = cw[1] * (v00 - v11) + v11;
01623
01624 w00 = cw[2] * (w00 - w01) + w01;
01625 w11 = cw[2] * (w10 - w11) + w11;
01626 *w = cw[1] * (w00 - w11) + w11;
01627 }
01628 #endif
01629

```



```

01630 /*****
01631
01632 void intpol_met_time_3d(
01633     met_t * met0,
01634     float array0[EX][EY][EP],
01635     met_t * met1,
01636     float array1[EX][EY][EP],
01637     double ts,
01638     double p,
01639     double lon,
01640     double lat,
01641     double *var,
01642     int *ci,
01643     double *cw,
01644     int init) {
01645
01646     double var0, var1, wt;
01647
01648     /* Spatial interpolation... */
01649     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01650     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01651
01652     /* Get weighting factor... */
01653     wt = (met1->time - ts) / (met1->time - met0->time);
01654
01655     /* Interpolate... */
01656     *var = wt * (var0 - var1) + var1;
01657 }
01658
01659 /*****
01660
01661 void intpol_met_time_2d(
01662     met_t * met0,
01663     float array0[EX][EY],
01664     met_t * met1,
01665     float array1[EX][EY],
01666     double ts,
01667     double lon,
01668     double lat,
01669     double *var,
01670     int *ci,
01671     double *cw,
01672     int init) {
01673
01674     double var0, var1, wt;
01675
01676     /* Spatial interpolation... */
01677     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01678     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01679
01680     /* Get weighting factor... */
01681     wt = (met1->time - ts) / (met1->time - met0->time);
01682
01683     /* Interpolate... */
01684     if (isfinite(var0) && isfinite(var1))
01685         *var = wt * (var0 - var1) + var1;
01686     else if (wt < 0.5)
01687         *var = var1;
01688     else
01689         *var = var0;
01690 }
01691
01692 /*****
01693
01694 #ifdef UVW
01695 void intpol_met_time_uvw(
01696     met_t * met0,
01697     met_t * met1,
01698     double ts,
01699     double p,
01700     double lon,
01701     double lat,
01702     double *u,
01703     double *v,
01704     double *w) {
01705
01706     double u0, u1, v0, v1, w0, w1, wt;
01707
01708     /* Spatial interpolation... */
01709     INTPOL_INIT;
01710     intpol_met_space_uvw(met0, p, lon, lat, &u0, &v0, &w0, ci, cw, 1);
01711     intpol_met_space_uvw(met1, p, lon, lat, &u1, &v1, &w1, ci, cw, 0);
01712
01713     /* Get weighting factor... */
01714     wt = (met1->time - ts) / (met1->time - met0->time);
01715
01716     /* Interpolate... */

```

```

01717     *u = wt * (u0 - u1) + u1;
01718     *v = wt * (v0 - v1) + v1;
01719     *w = wt * (w0 - w1) + w1;
01720 }
01721 #endif
01722
01723 /*****
01724
01725 void jsec2time(
01726     double jsec,
01727     int *year,
01728     int *mon,
01729     int *day,
01730     int *hour,
01731     int *min,
01732     int *sec,
01733     double *remain) {
01734
01735     struct tm t0, *t1;
01736
01737     t0.tm_year = 100;
01738     t0.tm_mon = 0;
01739     t0.tm_mday = 1;
01740     t0.tm_hour = 0;
01741     t0.tm_min = 0;
01742     t0.tm_sec = 0;
01743
01744     time_t jsec0 = (time_t) jsec + timegm(&t0);
01745     t1 = gmtime(&jsec0);
01746
01747     *year = t1->tm_year + 1900;
01748     *mon = t1->tm_mon + 1;
01749     *day = t1->tm_mday;
01750     *hour = t1->tm_hour;
01751     *min = t1->tm_min;
01752     *sec = t1->tm_sec;
01753     *remain = jsec - floor(jsec);
01754 }
01755
01756 /*****
01757
01758 double lapse_rate(
01759     double t,
01760     double h2o) {
01761
01762     /*
01763      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01764      and water vapor volume mixing ratio [1].
01765
01766      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01767      */
01768
01769     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01770
01771     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01772 }
01773
01774 /*****
01775
01776 int locate_irr(
01777     double *xx,
01778     int n,
01779     double x) {
01780
01781     int ilo = 0;
01782     int ihi = n - 1;
01783     int i = (ihi + ilo) >> 1;
01784
01785     if (xx[i] < xx[i + 1])
01786         while (ihi > ilo + 1) {
01787             i = (ihi + ilo) >> 1;
01788             if (xx[i] > x)
01789                 ihi = i;
01790             else
01791                 ilo = i;
01792         } else
01793         while (ihi > ilo + 1) {
01794             i = (ihi + ilo) >> 1;
01795             if (xx[i] <= x)
01796                 ihi = i;
01797             else
01798                 ilo = i;
01799         }
01800
01801     return ilo;
01802 }
01803

```

```

01804 /*****
01805
01806 int locate_reg(
01807     double *xx,
01808     int n,
01809     double x) {
01810
01811     /* Calculate index... */
01812     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01813
01814     /* Check range... */
01815     if (i < 0)
01816         return 0;
01817     else if (i > n - 2)
01818         return n - 2;
01819     else
01820         return i;
01821 }
01822
01823 /*****
01824
01825 double nat_temperature(
01826     double p,
01827     double h2o,
01828     double hno3) {
01829
01830     /* Check water vapor vmr... */
01831     h2o = GSL_MAX(h2o, 0.1e-6);
01832
01833     /* Calculate T_NAT... */
01834     double p_hno3 = hno3 * p / 1.333224;
01835     double p_h2o = h2o * p / 1.333224;
01836     double a = 0.009179 - 0.00088 * log10(p_h2o);
01837     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01838     double c = -11397.0 / a;
01839     double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
01840     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01841     if (x2 > 0)
01842         tnat = x2;
01843
01844     return tnat;
01845 }
01846
01847 /*****
01848
01849 void quicksort(
01850     double arr[],
01851     int brr[],
01852     int low,
01853     int high) {
01854
01855     if (low < high) {
01856         int pi = quicksort_partition(arr, brr, low, high);
01857
01858         #pragma omp task firstprivate(arr,brr,low,pi)
01859         {
01860             quicksort(arr, brr, low, pi - 1);
01861         }
01862
01863         // #pragma omp task firstprivate(arr,brr,high,pi)
01864         {
01865             quicksort(arr, brr, pi + 1, high);
01866         }
01867     }
01868 }
01869
01870 /*****
01871
01872 int quicksort_partition(
01873     double arr[],
01874     int brr[],
01875     int low,
01876     int high) {
01877
01878     double pivot = arr[high];
01879     int i = (low - 1);
01880
01881     for (int j = low; j <= high - 1; j++)
01882         if (arr[j] <= pivot) {
01883             i++;
01884             SWAP(arr[i], arr[j], double);
01885             SWAP(brr[i], brr[j], int);
01886         }
01887     SWAP(arr[high], arr[i + 1], double);
01888     SWAP(brr[high], brr[i + 1], int);
01889
01890     return (i + 1);

```

```

01891 }
01892
01893 /*****
01894
01895 int read_atm(
01896     const char *filename,
01897     ctl_t * ctl,
01898     atm_t * atm) {
01899
01900     int result;
01901
01902     /* Set timer... */
01903     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
01904
01905     /* Init... */
01906     atm->np = 0;
01907
01908     /* Write info... */
01909     LOG(1, "Read atmospheric data: %s", filename);
01910
01911     /* Read ASCII data... */
01912     if (ctl->atm_type == 0)
01913         result = read_atm_asc(filename, ctl, atm);
01914
01915     /* Read binary data... */
01916     else if (ctl->atm_type == 1)
01917         result = read_atm_bin(filename, ctl, atm);
01918
01919     /* Read netCDF data... */
01920     else if (ctl->atm_type == 2)
01921         result = read_atm_nc(filename, ctl, atm);
01922
01923     /* Read CLaMS data... */
01924     else if (ctl->atm_type == 3)
01925         result = read_atm_clams(filename, ctl, atm);
01926
01927     /* Error... */
01928     else
01929         ERRMSG("Atmospheric data type not supported!");
01930
01931     /* Check result... */
01932     if (result != 1)
01933         return 0;
01934
01935     /* Check number of air parcels... */
01936     if (atm->np < 1)
01937         ERRMSG("Can not read any data!");
01938
01939     /* Write info... */
01940     double mini, maxi;
01941     LOG(2, "Number of particles: %d", atm->np);
01942     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
01943     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
01944     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
01945     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
01946     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
01947     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
01948     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
01949     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
01950     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
01951     for (int iq = 0; iq < ctl->nq; iq++) {
01952         char msg[LEN];
01953         sprintf(msg, "Quantity %s range: %s ... %s %s",
01954             ctl->qnt_name[iq], ctl->qnt_format[iq],
01955             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
01956         gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
01957         LOG(2, msg, mini, maxi);
01958     }
01959
01960     /* Return success... */
01961     return 1;
01962 }
01963
01964 /*****
01965
01966 int read_atm_asc(
01967     const char *filename,
01968     ctl_t * ctl,
01969     atm_t * atm) {
01970
01971     FILE *in;
01972
01973     /* Open file... */
01974     if (!(in = fopen(filename, "r"))) {
01975         WARN("Cannot open file!");
01976         return 0;
01977     }

```

```

01978
01979 /* Read line... */
01980 char line[LEN];
01981 while (fgets(line, LEN, in)) {
01982
01983     /* Read data... */
01984     char *tok;
01985     TOK(line, tok, "%lg", atm->time[atm->np]);
01986     TOK(NULL, tok, "%lg", atm->p[atm->np]);
01987     TOK(NULL, tok, "%lg", atm->lon[atm->np]);
01988     TOK(NULL, tok, "%lg", atm->lat[atm->np]);
01989     for (int iq = 0; iq < ctl->nq; iq++)
01990         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
01991
01992     /* Convert altitude to pressure... */
01993     atm->p[atm->np] = P(atm->p[atm->np]);
01994
01995     /* Increment data point counter... */
01996     if ((++atm->np) > NP)
01997         ERRMSG("Too many data points!");
01998 }
01999
02000 /* Close file... */
02001 fclose(in);
02002
02003 /* Return success... */
02004 return 1;
02005 }
02006
02007 /*****
02008
02009 int read_atm_bin(
02010     const char *filename,
02011     ctl_t * ctl,
02012     atm_t * atm) {
02013
02014     FILE *in;
02015
02016     /* Open file... */
02017     if (!(in = fopen(filename, "r")))
02018         return 0;
02019
02020     /* Check version of binary data... */
02021     int version;
02022     FREAD(&version, int,
02023         1,
02024         in);
02025     if (version != 100)
02026         ERRMSG("Wrong version of binary data!");
02027
02028     /* Read data... */
02029     FREAD(&atm->np, int,
02030         1,
02031         in);
02032     FREAD(atm->time, double,
02033         (size_t) atm->np,
02034         in);
02035     FREAD(atm->p, double,
02036         (size_t) atm->np,
02037         in);
02038     FREAD(atm->lon, double,
02039         (size_t) atm->np,
02040         in);
02041     FREAD(atm->lat, double,
02042         (size_t) atm->np,
02043         in);
02044     for (int iq = 0; iq < ctl->nq; iq++)
02045         FREAD(atm->q[iq], double,
02046             (size_t) atm->np,
02047             in);
02048
02049     /* Read final flag... */
02050     int final;
02051     FREAD(&final, int,
02052         1,
02053         in);
02054     if (final != 999)
02055         ERRMSG("Error while reading binary data!");
02056
02057     /* Close file... */
02058     fclose(in);
02059
02060     /* Return success... */
02061     return 1;
02062 }
02063
02064 /*****

```

```

02065
02066 int read_atm_clams(
02067     const char *filename,
02068     ctl_t * ctl,
02069     atm_t * atm) {
02070
02071     int ncid, varid;
02072
02073     /* Open file... */
02074     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02075         return 0;
02076
02077     /* Get dimensions... */
02078     NC_INQ_DIM("NPARTS", &atm->np, 1, NP);
02079
02080     /* Get time... */
02081     if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
02082         NC_GET_VAR_DOUBLE(ncid, varid, atm->time);
02083     } else {
02084         WARN("TIME_INIT not found use time instead!");
02085         double time_init;
02086         NC_GET_DOUBLE("time", &time_init, 1);
02087         for (int ip = 0; ip < atm->np; ip++) {
02088             atm->time[ip] = time_init;
02089         }
02090     }
02091
02092     /* Read zeta coordinate, pressure is optional... */
02093     if (ctl->vert_coord_ap == 1) {
02094         NC_GET_DOUBLE("ZETA", atm->zeta, 1);
02095         NC_GET_DOUBLE("PRESS", atm->p, 0);
02096     }
02097
02098     /* Read pressure, zeta coordinate is optional... */
02099     else {
02100         NC_GET_DOUBLE("PRESS", atm->p, 1);
02101         NC_GET_DOUBLE("ZETA", atm->zeta, 0);
02102     }
02103
02104     /* Read longitude and latitude... */
02105     NC_GET_DOUBLE("LON", atm->lon, 1);
02106     NC_GET_DOUBLE("LAT", atm->lat, 1);
02107
02108     /* Close file... */
02109     NC(nc_close(ncid));
02110
02111     /* Return success... */
02112     return 1;
02113 }
02114
02115 /*****
02116
02117 int read_atm_nc(
02118     const char *filename,
02119     ctl_t * ctl,
02120     atm_t * atm) {
02121
02122     int ncid, varid;
02123
02124     /* Open file... */
02125     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02126         return 0;
02127
02128     /* Get dimensions... */
02129     NC_INQ_DIM("obs", &atm->np, 1, NP);
02130
02131     /* Read geolocations... */
02132     NC_GET_DOUBLE("time", atm->time, 1);
02133     NC_GET_DOUBLE("press", atm->p, 1);
02134     NC_GET_DOUBLE("lon", atm->lon, 1);
02135     NC_GET_DOUBLE("lat", atm->lat, 1);
02136
02137     /* Read variables... */
02138     for (int iq = 0; iq < ctl->nq; iq++)
02139         NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
02140
02141     /* Close file... */
02142     NC(nc_close(ncid));
02143
02144     /* Return success... */
02145     return 1;
02146 }
02147
02148 /*****
02149
02150 void read_clim(
02151     ctl_t * ctl,

```

```

02152     clim_t * clim) {
02153
02154     /* Set timer... */
02155     SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);
02156
02157     /* Init tropopause climatology... */
02158     clim_tropo_init(clim);
02159
02160     /* Init HNO3 climatology... */
02161     clim_hno3_init(clim);
02162
02163     /* Read OH climatology... */
02164     if (ctl->clim_oh_filename[0] != '-')
02165         clim_oh_init(ctl, clim);
02166
02167     /* Read H2O2 climatology... */
02168     if (ctl->clim_h2o2_filename[0] != '-')
02169         clim_h2o2_init(ctl, clim);
02170 }
02171
02172 /*****
02173
02174 void read_ctl(
02175     const char *filename,
02176     int argc,
02177     char *argv[],
02178     ctl_t * ctl) {
02179
02180     /* Set timer... */
02181     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02182
02183     /* Write info... */
02184     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02185         "(executable: %s | version: %s | compiled: %s, %s)\n",
02186         argv[0], VERSION, __DATE__, __TIME__);
02187
02188     /* Initialize quantity indices... */
02189     ctl->qnt_idx = -1;
02190     ctl->qnt_ens = -1;
02191     ctl->qnt_stat = -1;
02192     ctl->qnt_m = -1;
02193     ctl->qnt_vmr = -1;
02194     ctl->qnt_rp = -1;
02195     ctl->qnt_rhop = -1;
02196     ctl->qnt_ps = -1;
02197     ctl->qnt_ts = -1;
02198     ctl->qnt_zs = -1;
02199     ctl->qnt_us = -1;
02200     ctl->qnt_vs = -1;
02201     ctl->qnt_pbl = -1;
02202     ctl->qnt_pt = -1;
02203     ctl->qnt_tt = -1;
02204     ctl->qnt_zt = -1;
02205     ctl->qnt_h2ot = -1;
02206     ctl->qnt_z = -1;
02207     ctl->qnt_p = -1;
02208     ctl->qnt_t = -1;
02209     ctl->qnt_rho = -1;
02210     ctl->qnt_u = -1;
02211     ctl->qnt_v = -1;
02212     ctl->qnt_w = -1;
02213     ctl->qnt_h2o = -1;
02214     ctl->qnt_o3 = -1;
02215     ctl->qnt_lwc = -1;
02216     ctl->qnt_iwc = -1;
02217     ctl->qnt_pct = -1;
02218     ctl->qnt_pcb = -1;
02219     ctl->qnt_cl = -1;
02220     ctl->qnt_plcl = -1;
02221     ctl->qnt_plfc = -1;
02222     ctl->qnt_pel = -1;
02223     ctl->qnt_cape = -1;
02224     ctl->qnt_cin = -1;
02225     ctl->qnt_hno3 = -1;
02226     ctl->qnt_oh = -1;
02227     ctl->qnt_vmrimpl = -1;
02228     ctl->qnt_mloss_oh = -1;
02229     ctl->qnt_mloss_h2o2 = -1;
02230     ctl->qnt_mloss_wet = -1;
02231     ctl->qnt_mloss_dry = -1;
02232     ctl->qnt_mloss_decay = -1;
02233     ctl->qnt_psat = -1;
02234     ctl->qnt_psice = -1;
02235     ctl->qnt_pw = -1;
02236     ctl->qnt_sh = -1;
02237     ctl->qnt_rh = -1;
02238     ctl->qnt_rhice = -1;

```

```

02239   ctl->qnt_theta = -1;
02240   ctl->qnt_zeta = -1;
02241   ctl->qnt_tvirt = -1;
02242   ctl->qnt_lapse = -1;
02243   ctl->qnt_vh = -1;
02244   ctl->qnt_vz = -1;
02245   ctl->qnt_pv = -1;
02246   ctl->qnt_tdew = -1;
02247   ctl->qnt_tice = -1;
02248   ctl->qnt_tsts = -1;
02249   ctl->qnt_tnat = -1;
02250
02251   /* Read quantities... */
02252   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02253   if (ctl->nq > NQ)
02254       ERRMSG("Too many quantities!");
02255   for (int iq = 0; iq < ctl->nq; iq++) {
02256
02257       /* Read quantity name and format... */
02258       scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02259       scan_ctl(filename, argc, argv, "QNT_LONGNAME", iq, ctl->qnt_name[iq],
02260               ctl->qnt_longname[iq]);
02261       scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02262               ctl->qnt_format[iq]);
02263
02264       /* Try to identify quantity... */
02265       SET_QNT(qnt_idx, "idx", "particle index", "-")
02266       SET_QNT(qnt_ens, "ens", "ensemble index", "-")
02267       SET_QNT(qnt_stat, "stat", "station flag", "-")
02268       SET_QNT(qnt_m, "m", "mass", "kg")
02269       SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
02270       SET_QNT(qnt_rp, "rp", "particle radius", "microns")
02271       SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
02272       SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
02273       SET_QNT(qnt_ts, "ts", "surface temperature", "K")
02274       SET_QNT(qnt_zs, "zs", "surface height", "km")
02275       SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
02276       SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
02277       SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
02278       SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
02279       SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
02280       SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
02281       SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
02282       SET_QNT(qnt_z, "z", "geopotential height", "km")
02283       SET_QNT(qnt_p, "p", "pressure", "hPa")
02284       SET_QNT(qnt_t, "t", "temperature", "K")
02285       SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
02286       SET_QNT(qnt_u, "u", "zonal wind", "m/s")
02287       SET_QNT(qnt_v, "v", "meridional wind", "m/s")
02288       SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
02289       SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
02290       SET_QNT(qnt_o3, "o3", "ozone", "ppv")
02291       SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
02292       SET_QNT(qnt_lwc, "lwc", "cloud liquid water content", "kg/kg")
02293       SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
02294       SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
02295       SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
02296       SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
02297       SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
02298       SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
02299       SET_QNT(qnt_cape, "cape", "convective available potential energy",
02300               "J/kg")
02301       SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
02302       SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
02303       SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
02304       SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
02305       SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
02306       SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
02307               "kg")
02308       SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
02309               "kg")
02310       SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
02311               "kg")
02312       SET_QNT(qnt_mloss_decay, "mloss_decay",
02313               "mass loss due to exponential decay", "kg")
02314       SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
02315       SET_QNT(qnt_psize, "psize", "saturation pressure over ice", "hPa")
02316       SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
02317       SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
02318       SET_QNT(qnt_rh, "rh", "relative humidity", "%")
02319       SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%")
02320       SET_QNT(qnt_theta, "theta", "potential temperature", "K")
02321       SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
02322       SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
02323       SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
02324       SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
02325       SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")

```



```

02326     SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
02327     SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
02328     SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
02329     SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
02330     SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
02331     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02332 }
02333
02334 /* netCDF I/O parameters... */
02335 ctl->chunkszhint =
02336     (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02337     NULL);
02338 ctl->read_mode =
02339     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02340
02341 /* Vertical coordinates and velocities... */
02342 ctl->vert_coord_ap =
02343     (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
02344 ctl->vert_coord_met =
02345     (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
02346 ctl->vert_vel =
02347     (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
02348 ctl->clams_met_data =
02349     (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);
02350
02351 /* Time steps of simulation... */
02352 ctl->direction =
02353     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02354 if (ctl->direction != -1 && ctl->direction != 1)
02355     ERRMSG("Set DIRECTION to -1 or 1!");
02356 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02357 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02358
02359 /* Meteo data... */
02360 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02361 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02362 ctl->met_type =
02363     (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
02364 ctl->met_nc_scale =
02365     (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", NULL);
02366 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02367 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02368 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02369 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02370     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02371 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02372 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02373 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02374 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02375     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02376 ctl->met_detrend =
02377     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02378 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02379 if (ctl->met_np > EP)
02380     ERRMSG("Too many levels!");
02381 for (int ip = 0; ip < ctl->met_np; ip++)
02382     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02383 ctl->met_geopot_sx
02384     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02385 ctl->met_geopot_sy
02386     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02387 ctl->met_tropo =
02388     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02389 if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02390     ERRMSG("Set MET_TROPO = 0 ... 5!");
02391 ctl->met_tropo_lapse =
02392     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02393 ctl->met_tropo_nlev =
02394     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02395 ctl->met_tropo_lapse_sep =
02396     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02397 ctl->met_tropo_nlev_sep =
02398     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02399     NULL);
02400 ctl->met_tropo_pv =
02401     scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02402 ctl->met_tropo_theta =
02403     scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02404 ctl->met_tropo_spline =
02405     (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02406 ctl->met_cloud =
02407     (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02408 if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02409     ERRMSG("Set MET_CLOUD = 0 ... 3!");
02410 ctl->met_cloud_min =
02411     scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
02412 ctl->met_dt_out =

```

```

02413     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02414     ctl->met_cache =
02415         (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02416
02417     /* Sorting... */
02418     ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);
02419
02420     /* Isosurface parameters... */
02421     ctl->isosurf =
02422         (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02423     scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02424
02425     /* Advection parameters... */
02426     ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
02427     if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
02428         ERRMSG("Set ADVECT to 1, 2, or 4!");
02429     ctl->reflect =
02430         (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02431
02432     /* Diffusion parameters... */
02433     ctl->turb_dx_trop =
02434         scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02435     ctl->turb_dx_strat =
02436         scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02437     ctl->turb_dz_trop =
02438         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02439     ctl->turb_dz_strat =
02440         scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02441     ctl->turb_mesox =
02442         scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02443     ctl->turb_mesoz =
02444         scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02445
02446     /* Convection... */
02447     ctl->conv_cape =
02448         scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02449     ctl->conv_cin =
02450         scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02451     ctl->conv_wmax =
02452         scan_ctl(filename, argc, argv, "CONV_WMAX", -1, "-999", NULL);
02453     ctl->conv_wcape =
02454         (int) scan_ctl(filename, argc, argv, "CONV_WCAPE", -1, "0", NULL);
02455     ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02456     ctl->conv_mix =
02457         (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "0", NULL);
02458     ctl->conv_mix_bot =
02459         (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02460     ctl->conv_mix_top =
02461         (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02462
02463     /* Boundary conditions... */
02464     ctl->bound_mass =
02465         scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02466     ctl->bound_mass_trend =
02467         scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
02468     ctl->bound_vmr =
02469         scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02470     ctl->bound_vmr_trend =
02471         scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
02472     ctl->bound_lat0 =
02473         scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02474     ctl->bound_lat1 =
02475         scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02476     ctl->bound_p0 =
02477         scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02478     ctl->bound_p1 =
02479         scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02480     ctl->bound_dps =
02481         scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02482
02483     /* Species parameters... */
02484     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02485     if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02486         ctl->molmass = 120.907;
02487         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
02488         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
02489     } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02490         ctl->molmass = 137.359;
02491         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
02492         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
02493     } else if (strcasecmp(ctl->species, "CH4") == 0) {
02494         ctl->molmass = 16.043;
02495         ctl->oh_chem_reaction = 2;
02496         ctl->oh_chem[0] = 2.45e-12;
02497         ctl->oh_chem[1] = 1775;
02498         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
02499         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;

```

```
02500 } else if (strcasecmp(ctl->species, "CO") == 0) {
02501     ctl->molmass = 28.01;
02502     ctl->oh_chem_reaction = 3;
02503     ctl->oh_chem[0] = 6.9e-33;
02504     ctl->oh_chem[1] = 2.1;
02505     ctl->oh_chem[2] = 1.1e-12;
02506     ctl->oh_chem[3] = -1.3;
02507     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
02508     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
02509 } else if (strcasecmp(ctl->species, "CO2") == 0) {
02510     ctl->molmass = 44.009;
02511     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
02512     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02513 } else if (strcasecmp(ctl->species, "H2O") == 0) {
02514     ctl->molmass = 18.01528;
02515 } else if (strcasecmp(ctl->species, "N2O") == 0) {
02516     ctl->molmass = 44.013;
02517     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
02518     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
02519 } else if (strcasecmp(ctl->species, "NH3") == 0) {
02520     ctl->molmass = 17.031;
02521     ctl->oh_chem_reaction = 2;
02522     ctl->oh_chem[0] = 1.7e-12;
02523     ctl->oh_chem[1] = 710;
02524     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
02525     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
02526 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02527     ctl->molmass = 63.012;
02528     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
02529     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
02530 } else if (strcasecmp(ctl->species, "NO") == 0) {
02531     ctl->molmass = 30.006;
02532     ctl->oh_chem_reaction = 3;
02533     ctl->oh_chem[0] = 7.1e-31;
02534     ctl->oh_chem[1] = 2.6;
02535     ctl->oh_chem[2] = 3.6e-11;
02536     ctl->oh_chem[3] = 0.1;
02537     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
02538     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02539 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02540     ctl->molmass = 46.005;
02541     ctl->oh_chem_reaction = 3;
02542     ctl->oh_chem[0] = 1.8e-30;
02543     ctl->oh_chem[1] = 3.0;
02544     ctl->oh_chem[2] = 2.8e-11;
02545     ctl->oh_chem[3] = 0.0;
02546     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
02547     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02548 } else if (strcasecmp(ctl->species, "O3") == 0) {
02549     ctl->molmass = 47.997;
02550     ctl->oh_chem_reaction = 2;
02551     ctl->oh_chem[0] = 1.7e-12;
02552     ctl->oh_chem[1] = 940;
02553     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
02554     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
02555 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02556     ctl->molmass = 146.048;
02557     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
02558     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
02559 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02560     ctl->molmass = 64.066;
02561     ctl->oh_chem_reaction = 3;
02562     ctl->oh_chem[0] = 2.9e-31;
02563     ctl->oh_chem[1] = 4.1;
02564     ctl->oh_chem[2] = 1.7e-12;
02565     ctl->oh_chem[3] = -0.2;
02566     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
02567     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
02568 } else {
02569     ctl->molmass =
02570         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02571     ctl->oh_chem_reaction =
02572         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02573     ctl->h2o2_chem_reaction =
02574         (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
02575             NULL);
02576     for (int ip = 0; ip < 4; ip++)
02577         ctl->oh_chem[ip] =
02578             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02579     for (int ip = 0; ip < 1; ip++)
02580         ctl->dry_depo[ip] =
02581             scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02582     ctl->wet_depo_ic_a =
02583         scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
02584     ctl->wet_depo_ic_b =
02585         scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
02586     ctl->wet_depo_bc_a =
```

```

02587     scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
02588     ctl->wet_depo_bc_b =
02589     scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
02590     for (int ip = 0; ip < 3; ip++)
02591         ctl->wet_depo_ic_h[ip] =
02592         scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
02593     for (int ip = 0; ip < 1; ip++)
02594         ctl->wet_depo_bc_h[ip] =
02595         scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
02596 }
02597
02598 /* Wet deposition... */
02599 ctl->wet_depo_pre[0] =
02600     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
02601     ctl->wet_depo_pre[1] =
02602     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
02603     ctl->wet_depo_ic_ret_ratio =
02604     scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
02605     ctl->wet_depo_bc_ret_ratio =
02606     scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);
02607
02608 /* OH chemistry... */
02609 ctl->oh_chem_beta =
02610     scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
02611     scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
02612         ".../data/clams_radical_species.nc", ctl->clim_oh_filename);
02613
02614 /* H2O2 chemistry... */
02615     ctl->h2o2_chem_cc =
02616     scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
02617     scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
02618         ".../data/cams_H2O2.nc", ctl->clim_h2o2_filename);
02619
02620 /* Exponential decay... */
02621     ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02622     ctl->tdec_strat =
02623     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02624
02625 /* PSC analysis... */
02626     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02627     ctl->psc_hno3 =
02628     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02629
02630 /* Output of atmospheric data... */
02631     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02632     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02633     ctl->atm_dt_out =
02634     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02635     ctl->atm_filter =
02636     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02637     ctl->atm_stride =
02638     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02639     ctl->atm_type =
02640     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02641
02642 /* Output of CSI data... */
02643     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02644     ctl->csi_dt_out =
02645     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02646     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02647     ctl->csi_obsmin =
02648     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02649     ctl->csi_modmin =
02650     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02651     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02652     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02653     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02654     ctl->csi_lon0 =
02655     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02656     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02657     ctl->csi_nx =
02658     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02659     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02660     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02661     ctl->csi_ny =
02662     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02663
02664 /* Output of ensemble data... */
02665     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02666     ctl->ens_dt_out =
02667     scan_ctl(filename, argc, argv, "ENS_DT_OUT", -1, "86400", NULL);
02668
02669 /* Output of grid data... */
02670     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02671         ctl->grid_basename);
02672     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02673     ctl->grid_dt_out =

```

```

02674     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02675     ctl->grid_sparse =
02676         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02677     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02678     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02679     ctl->grid_nz =
02680         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02681     ctl->grid_lon0 =
02682         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02683     ctl->grid_lon1 =
02684         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02685     ctl->grid_nx =
02686         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02687     ctl->grid_lat0 =
02688         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02689     ctl->grid_lat1 =
02690         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02691     ctl->grid_ny =
02692         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02693     ctl->grid_type =
02694         (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", NULL);
02695
02696     /* Output of profile data... */
02697     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02698         ctl->prof_basename);
02699     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02700     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02701     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02702     ctl->prof_nz =
02703         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02704     ctl->prof_lon0 =
02705         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02706     ctl->prof_lon1 =
02707         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02708     ctl->prof_nx =
02709         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02710     ctl->prof_lat0 =
02711         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02712     ctl->prof_lat1 =
02713         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02714     ctl->prof_ny =
02715         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02716
02717     /* Output of sample data... */
02718     scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02719         ctl->sample_basename);
02720     scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02721         ctl->sample_obsfile);
02722     ctl->sample_dx =
02723         scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02724     ctl->sample_dz =
02725         scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02726
02727     /* Output of station data... */
02728     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02729         ctl->stat_basename);
02730     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02731     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02732     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02733     ctl->stat_t0 =
02734         scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02735     ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02736 }
02737
02738 /******
02739
02740 int read_met(
02741     char *filename,
02742     ctl_t * ctl,
02743     clim_t * clim,
02744     met_t * met) {
02745
02746     /* Write info... */
02747     LOG(1, "Read meteo data: %s", filename);
02748
02749     /* Read netCDF data... */
02750     if (ctl->met_type == 0) {
02751
02752         int ncid;
02753
02754         /* Open netCDF file... */
02755         if (nc_open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02756             NC_NOERR) {
02757             WARN("Cannot open file!");
02758             return 0;
02759         }
02760

```

```

02761      /* Read coordinates of meteo data... */
02762      read_met_grid(filename, ncid, ctl, met);
02763
02764      /* Read meteo data on vertical levels... */
02765      read_met_levels(ncid, ctl, met);
02766
02767      /* Extrapolate data for lower boundary... */
02768      read_met_extrapolate(met);
02769
02770      /* Read surface data... */
02771      read_met_surface(ncid, met, ctl);
02772
02773      /* Create periodic boundary conditions... */
02774      read_met_periodic(met);
02775
02776      /* Downsampling... */
02777      read_met_sample(ctl, met);
02778
02779      /* Calculate geopotential heights... */
02780      read_met_geopot(ctl, met);
02781
02782      /* Calculate potential vorticity... */
02783      read_met_pv(met);
02784
02785      /* Calculate boundary layer data... */
02786      read_met_pbl(met);
02787
02788      /* Calculate tropopause data... */
02789      read_met_tropo(ctl, clim, met);
02790
02791      /* Calculate cloud properties... */
02792      read_met_cloud(ctl, met);
02793
02794      /* Calculate convective available potential energy... */
02795      read_metCAPE(clim, met);
02796
02797      /* Detrending... */
02798      read_met_detrend(ctl, met);
02799
02800      /* Close file... */
02801      NC(nc_close(ncid));
02802  }
02803
02804  /* Read binary data... */
02805  else if (ctl->met_type >= 1 && ctl->met_type <= 4) {
02806
02807      FILE *in;
02808
02809      double r;
02810
02811      int year, mon, day, hour, min, sec;
02812
02813      /* Set timer... */
02814      SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);
02815
02816      /* Open file... */
02817      if (!(in = fopen(filename, "r"))) {
02818          WARN("Cannot open file!");
02819          return 0;
02820      }
02821
02822      /* Check type of binary data... */
02823      int met_type;
02824      FREAD(&met_type, int,
02825          1,
02826          in);
02827      if (met_type != ctl->met_type)
02828          ERRMSG("Wrong MET_TYPE of binary data!");
02829
02830      /* Check version of binary data... */
02831      int version;
02832      FREAD(&version, int,
02833          1,
02834          in);
02835      if (version != 100)
02836          ERRMSG("Wrong version of binary data!");
02837
02838      /* Read time... */
02839      FREAD(&met->time, double,
02840          1,
02841          in);
02842      jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
02843      LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
02844          met->time, year, mon, day, hour, min);
02845      if (year < 1900 || year > 2100 || mon < 1 || mon > 12
02846          || day < 1 || day > 31 || hour < 0 || hour > 23)
02847          ERRMSG("Error while reading time!");

```

```

02848
02849  /* Read dimensions... */
02850  FREAD(&met->nx, int,
02851      1,
02852      in);
02853  LOG(2, "Number of longitudes: %d", met->nx);
02854  if (met->nx < 2 || met->nx > EX)
02855      ERRMSG("Number of longitudes out of range!");
02856
02857  FREAD(&met->ny, int,
02858      1,
02859      in);
02860  LOG(2, "Number of latitudes: %d", met->ny);
02861  if (met->ny < 2 || met->ny > EY)
02862      ERRMSG("Number of latitudes out of range!");
02863
02864  FREAD(&met->np, int,
02865      1,
02866      in);
02867  LOG(2, "Number of levels: %d", met->np);
02868  if (met->np < 2 || met->np > EP)
02869      ERRMSG("Number of levels out of range!");
02870
02871  /* Read grid... */
02872  FREAD(met->lon, double,
02873      (size_t) met->nx,
02874      in);
02875  LOG(2, "Longitudes: %g, %g ... %g deg",
02876      met->lon[0], met->lon[1], met->lon[met->nx - 1]);
02877
02878  FREAD(met->lat, double,
02879      (size_t) met->ny,
02880      in);
02881  LOG(2, "Latitudes: %g, %g ... %g deg",
02882      met->lat[0], met->lat[1], met->lat[met->ny - 1]);
02883
02884  FREAD(met->p, double,
02885      (size_t) met->np,
02886      in);
02887  LOG(2, "Altitude levels: %g, %g ... %g km",
02888      Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
02889  LOG(2, "Pressure levels: %g, %g ... %g hPa",
02890      met->p[0], met->p[1], met->p[met->np - 1]);
02891
02892  /* Read surface data... */
02893  read_met_bin_2d(in, met, met->ps, "PS");
02894  read_met_bin_2d(in, met, met->ts, "TS");
02895  read_met_bin_2d(in, met, met->zs, "ZS");
02896  read_met_bin_2d(in, met, met->us, "US");
02897  read_met_bin_2d(in, met, met->vs, "VS");
02898  read_met_bin_2d(in, met, met->pbl, "PBL");
02899  read_met_bin_2d(in, met, met->pt, "PT");
02900  read_met_bin_2d(in, met, met->tt, "TT");
02901  read_met_bin_2d(in, met, met->zt, "ZT");
02902  read_met_bin_2d(in, met, met->h2ot, "H2OT");
02903  read_met_bin_2d(in, met, met->pct, "PCT");
02904  read_met_bin_2d(in, met, met->pcb, "PCB");
02905  read_met_bin_2d(in, met, met->cl, "CL");
02906  read_met_bin_2d(in, met, met->plcl, "PLCL");
02907  read_met_bin_2d(in, met, met->plfc, "PLFC");
02908  read_met_bin_2d(in, met, met->pel, "PEL");
02909  read_met_bin_2d(in, met, met->cape, "CAPE");
02910  read_met_bin_2d(in, met, met->cin, "CIN");
02911
02912  /* Read level data... */
02913  read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
02914  read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
02915  read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
02916  read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
02917  read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
02918  read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
02919  read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
02920  read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
02921  read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
02922  read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);
02923
02924  /* Read final flag... */
02925  int final;
02926  FREAD(&final, int,
02927      1,
02928      in);
02929  if (final != 999)
02930      ERRMSG("Error while reading binary data!");
02931
02932  /* Close file... */
02933  fclose(in);
02934  }

```

```

02935
02936 /* Not implemented... */
02937 else
02938     ERRMSG("MET_TYPE not implemented!");
02939
02940 /* Copy wind data to cache... */
02941 #ifdef UVW
02942 #pragma omp parallel for default(shared) collapse(2)
02943 for (int ix = 0; ix < met->nx; ix++)
02944     for (int iy = 0; iy < met->ny; iy++)
02945         for (int ip = 0; ip < met->np; ip++) {
02946             met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
02947             met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
02948             met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
02949         }
02950 #endif
02951
02952 /* Return success... */
02953 return 1;
02954 }
02955
02956 /*****
02957 void read_met_bin_2d(
02958     FILE * in,
02959     met_t * met,
02960     float var[EX][EY],
02961     char *varname) {
02962
02963     float *help;
02964
02965     /* Allocate... */
02966     ALLOC(help, float,
02967           EX * EY);
02969
02970     /* Read uncompressed... */
02971     LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
02972     FREAD(help, float,
02973           (size_t) (met->nx * met->ny),
02974           in);
02975
02976     /* Copy data... */
02977     for (int ix = 0; ix < met->nx; ix++)
02978         for (int iy = 0; iy < met->ny; iy++)
02979             var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];
02980
02981     /* Free... */
02982     free(help);
02983 }
02984
02985 /*****
02986 void read_met_bin_3d(
02987     FILE * in,
02988     ctl_t * ctl,
02989     met_t * met,
02990     float var[EX][EY][EP],
02991     char *varname,
02992     int precision,
02993     double tolerance) {
02994
02995     float *help;
02996
02997     /* Allocate... */
02998     ALLOC(help, float,
02999           EX * EY * EP);
03000
03001     /* Read uncompressed data... */
03002     if (ctl->met_type == 1) {
03003         LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
03004         FREAD(help, float,
03005               (size_t) (met->nx * met->ny * met->np),
03006               in);
03007     }
03008
03009     /* Read packed data... */
03010     else if (ctl->met_type == 2)
03011         compress_pack(varname, help, (size_t) (met->ny * met->nx),
03012                       (size_t) met->np, 1, in);
03013
03014     /* Read zfp data... */
03015     else if (ctl->met_type == 3) {
03016         #ifdef ZFP
03017             compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
03018                           tolerance, 1, in);
03019         #else
03020             ERRMSG("zfp compression not supported!");
03021

```



```

03022     LOG(3, "%d %g", precision, tolerance);
03023 #endif
03024 }
03025
03026 /* Read zstd data... */
03027 else if (ctl->met_type == 4) {
03028 #ifndef ZSTD
03029     compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
03030                 in);
03031 #else
03032     ERRMSG("zstd compression not supported!");
03033 #endif
03034 }
03035
03036 /* Copy data... */
03037 #pragma omp parallel for default(shared) collapse(2)
03038 for (int ix = 0; ix < met->nx; ix++)
03039     for (int iy = 0; iy < met->ny; iy++)
03040         for (int ip = 0; ip < met->np; ip++)
03041             var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];
03042
03043 /* Free... */
03044 free(help);
03045 }
03046
03047 /*****
03048
03049 void read_met_cape(
03050     clim_t * clim,
03051     met_t * met) {
03052
03053     /* Set timer... */
03054     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
03055     LOG(2, "Calculate CAPE...");
03056
03057     /* Vertical spacing (about 100 m)... */
03058     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
03059
03060     /* Loop over columns... */
03061 #pragma omp parallel for default(shared) collapse(2)
03062 for (int ix = 0; ix < met->nx; ix++)
03063     for (int iy = 0; iy < met->ny; iy++) {
03064
03065         /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
03066         int n = 0;
03067         double h2o = 0, t, theta = 0;
03068         double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
03069         double ptop = pbot - 50.;
03070         for (int ip = 0; ip < met->np; ip++) {
03071             if (met->p[ip] <= pbot) {
03072                 theta += THETA(met->p[ip], met->t[ix][iy][ip]);
03073                 h2o += met->h2o[ix][iy][ip];
03074                 n++;
03075             }
03076             if (met->p[ip] < ptop && n > 0)
03077                 break;
03078         }
03079         theta /= n;
03080         h2o /= n;
03081
03082         /* Cannot compute anything if water vapor is missing... */
03083         met->plcl[ix][iy] = GSL_NAN;
03084         met->plfc[ix][iy] = GSL_NAN;
03085         met->pel[ix][iy] = GSL_NAN;
03086         met->cape[ix][iy] = GSL_NAN;
03087         met->cin[ix][iy] = GSL_NAN;
03088         if (h2o <= 0)
03089             continue;
03090
03091         /* Find lifted condensation level (LCL)... */
03092         ptop = P(20.);
03093         pbot = met->ps[ix][iy];
03094         do {
03095             met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
03096             t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
03097             if (RH(met->plcl[ix][iy], t, h2o) > 100.)
03098                 ptop = met->plcl[ix][iy];
03099             else
03100                 pbot = met->plcl[ix][iy];
03101         } while (pbot - ptop > 0.1);
03102
03103         /* Calculate CIN up to LCL... */
03104         INTPOL_INIT;
03105         double dcape, dz, h2o_env, t_env;
03106         double p = met->ps[ix][iy];
03107         met->cape[ix][iy] = met->cin[ix][iy] = 0;
03108         do {

```

```

03109     dz = dz0 * TVIRT(t, h2o);
03110     p /= pfac;
03111     t = theta / pow(1000. / p, 0.286);
03112     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03113                        &t_env, ci, cw, 1);
03114     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03115                        &h2o_env, ci, cw, 0);
03116     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03117             TVIRT(t_env, h2o_env) * dz;
03118     if (dcape < 0)
03119         met->cin[ix][iy] += fabsf((float) dcape);
03120 } while (p > met->plcl[ix][iy]);
03121
03122 /* Calculate level of free convection (LFC), equilibrium level (EL),
03123    and convective available potential energy (CAPE)... */
03124 dcape = 0;
03125 p = met->plcl[ix][iy];
03126 t = theta / pow(1000. / p, 0.286);
03127 ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
03128 do {
03129     dz = dz0 * TVIRT(t, h2o);
03130     p /= pfac;
03131     t -= lapse_rate(t, h2o) * dz;
03132     double psat = PSAT(t);
03133     h2o = psat / (p - (1. - EPS) * psat);
03134     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03135                        &t_env, ci, cw, 1);
03136     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03137                        &h2o_env, ci, cw, 0);
03138     double dcape_old = dcape;
03139     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03140             TVIRT(t_env, h2o_env) * dz;
03141     if (dcape > 0) {
03142         met->cape[ix][iy] += (float) dcape;
03143         if (!isfinite(met->plfc[ix][iy]))
03144             met->plfc[ix][iy] = (float) p;
03145     } else if (dcape_old > 0)
03146         met->pel[ix][iy] = (float) p;
03147     if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
03148         met->cin[ix][iy] += fabsf((float) dcape);
03149 } while (p > ptop);
03150
03151 /* Check results... */
03152 if (!isfinite(met->plfc[ix][iy]))
03153     met->cin[ix][iy] = GSL_NAN;
03154 }
03155 }
03156
03157 /*****
03158 void read_met_cloud(
03159     ctl_t * ctl,
03160     met_t * met) {
03161
03162     /* Set timer... */
03163     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
03164     LOG(2, "Calculate cloud data...");
03165
03166     /* Loop over columns... */
03167     #pragma omp parallel for default(shared) collapse(2)
03168     for (int ix = 0; ix < met->nx; ix++)
03169         for (int iy = 0; iy < met->ny; iy++) {
03170
03171             /* Init... */
03172             met->pct[ix][iy] = GSL_NAN;
03173             met->pcb[ix][iy] = GSL_NAN;
03174             met->cl[ix][iy] = 0;
03175
03176             /* Loop over pressure levels... */
03177             for (int ip = 0; ip < met->np - 1; ip++) {
03178
03179                 /* Check pressure... */
03180                 if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
03181                     continue;
03182
03183                 /* Check ice water and liquid water content... */
03184                 if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
03185                     || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {
03186
03187                     /* Get cloud top pressure ... */
03188                     met->pct[ix][iy]
03189                         = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
03190
03191                     /* Get cloud bottom pressure ... */
03192                     if (!isfinite(met->pcb[ix][iy]))
03193                         met->pcb[ix][iy]
03194                             = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
03195

```

```

03196     }
03197
03198     /* Get cloud water... */
03199     met->cl[ix][iy] += (float)
03200         (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
03201             + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
03202          * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
03203     }
03204 }
03205 }
03206
03207 /*****
03208 void read_met_detrend(
03209     ctl_t * ctl,
03210     met_t * met) {
03211     met_t *help;
03212
03213     /* Check parameters... */
03214     if (ctl->met_detrend <= 0)
03215         return;
03216
03217     /* Set timer... */
03218     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
03219     LOG(2, "Detrend meteo data...");
03220
03221     /* Allocate... */
03222     ALLOC(help, met_t, 1);
03223
03224     /* Calculate standard deviation... */
03225     double sigma = ctl->met_detrend / 2.355;
03226     double tssq = 2. * SQR(sigma);
03227
03228     /* Calculate box size in latitude... */
03229     int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03230     sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03231
03232     /* Calculate background... */
03233     #pragma omp parallel for default(shared) collapse(2)
03234     for (int ix = 0; ix < met->nx; ix++) {
03235         for (int iy = 0; iy < met->ny; iy++) {
03236
03237             /* Calculate Cartesian coordinates... */
03238             double x0[3];
03239             geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03240
03241             /* Calculate box size in longitude... */
03242             int sx =
03243                 (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03244                     fabs(met->lon[1] - met->lon[0]));
03245             sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03246
03247             /* Init... */
03248             float wsum = 0;
03249             for (int ip = 0; ip < met->np; ip++) {
03250                 help->t[ix][iy][ip] = 0;
03251                 help->u[ix][iy][ip] = 0;
03252                 help->v[ix][iy][ip] = 0;
03253                 help->w[ix][iy][ip] = 0;
03254             }
03255
03256             /* Loop over neighboring grid points... */
03257             for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03258                 int ix3 = ix2;
03259                 if (ix3 < 0)
03260                     ix3 += met->nx;
03261                 else if (ix3 >= met->nx)
03262                     ix3 -= met->nx;
03263                 for (int iy2 = GSL_MAX(iy - sy, 0);
03264                     iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03265
03266                     /* Calculate Cartesian coordinates... */
03267                     double x1[3];
03268                     geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03269
03270                     /* Calculate weighting factor... */
03271                     float w = (float) exp(-DIST2(x0, x1) / tssq);
03272
03273                     /* Add data... */
03274                     wsum += w;
03275                     for (int ip = 0; ip < met->np; ip++) {
03276                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03277                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03278                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03279                         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03280                     }
03281                 }
03282             }

```

```

03283     }
03284 }
03285
03286 /* Normalize... */
03287 for (int ip = 0; ip < met->np; ip++) {
03288     help->t[ix][iy][ip] /= wsum;
03289     help->u[ix][iy][ip] /= wsum;
03290     help->v[ix][iy][ip] /= wsum;
03291     help->w[ix][iy][ip] /= wsum;
03292 }
03293 }
03294 }
03295
03296 /* Subtract background... */
03297 #pragma omp parallel for default(shared) collapse(3)
03298 for (int ix = 0; ix < met->nx; ix++)
03299     for (int iy = 0; iy < met->ny; iy++)
03300         for (int ip = 0; ip < met->np; ip++) {
03301             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03302             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03303             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03304             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03305         }
03306
03307 /* Free... */
03308 free(help);
03309 }
03310
03311 /*****
03312
03313 void read_met_extrapolate(
03314     met_t * met) {
03315
03316     /* Set timer... */
03317     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03318     LOG(2, "Extrapolate meteo data...");
03319
03320     /* Loop over columns... */
03321     #pragma omp parallel for default(shared) collapse(2)
03322     for (int ix = 0; ix < met->nx; ix++)
03323         for (int iy = 0; iy < met->ny; iy++) {
03324
03325             /* Find lowest valid data point... */
03326             int ip0;
03327             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03328                 if (!isfinite(met->t[ix][iy][ip0])
03329                     || !isfinite(met->u[ix][iy][ip0])
03330                     || !isfinite(met->v[ix][iy][ip0])
03331                     || !isfinite(met->w[ix][iy][ip0]))
03332                     break;
03333
03334             /* Extrapolate... */
03335             for (int ip = ip0; ip >= 0; ip--) {
03336                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03337                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03338                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03339                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03340                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03341                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03342                 met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03343                 met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03344             }
03345         }
03346 }
03347
03348 /*****
03349
03350 void read_met_geopot(
03351     ctl_t * ctl,
03352     met_t * met) {
03353
03354     static float help[EP][EX][EY];
03355
03356     double logp[EP];
03357
03358     int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03359
03360     /* Set timer... */
03361     SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03362     LOG(2, "Calculate geopotential heights...");
03363
03364     /* Calculate log pressure... */
03365     #pragma omp parallel for default(shared)
03366     for (int ip = 0; ip < met->np; ip++)
03367         logp[ip] = log(met->p[ip]);
03368
03369     /* Apply hydrostatic equation to calculate geopotential heights... */

```

```

03370 #pragma omp parallel for default(shared) collapse(2)
03371     for (int ix = 0; ix < met->nx; ix++)
03372         for (int iy = 0; iy < met->ny; iy++) {
03373
03374             /* Get surface height and pressure... */
03375             double zs = met->zs[ix][iy];
03376             double lnps = log(met->ps[ix][iy]);
03377
03378             /* Get temperature and water vapor vmr at the surface... */
03379             int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03380             double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03381                             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03382             double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03383                               met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03384
03385             /* Upper part of profile... */
03386             met->z[ix][iy][ip0 + 1]
03387                 = (float) (zs +
03388                           ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03389                                met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03390             for (int ip = ip0 + 2; ip < met->np; ip++)
03391                 met->z[ix][iy][ip]
03392                     = (float) (met->z[ix][iy][ip - 1] +
03393                               ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03394                                    met->h2o[ix][iy][ip - 1], logp[ip],
03395                                    met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03396
03397             /* Lower part of profile... */
03398             met->z[ix][iy][ip0]
03399                 = (float) (zs +
03400                           ZDIFF(lnps, ts, h2os, logp[ip0],
03401                                met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03402             for (int ip = ip0 - 1; ip >= 0; ip--)
03403                 met->z[ix][iy][ip]
03404                     = (float) (met->z[ix][iy][ip + 1] +
03405                               ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03406                                    met->h2o[ix][iy][ip + 1], logp[ip],
03407                                    met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03408         }
03409
03410     /* Check control parameters... */
03411     if (dx == 0 || dy == 0)
03412         return;
03413
03414     /* Default smoothing parameters... */
03415     if (dx < 0 || dy < 0) {
03416         if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03417             dx = 3;
03418             dy = 2;
03419         } else {
03420             dx = 6;
03421             dy = 4;
03422         }
03423     }
03424
03425     /* Calculate weights for smoothing... */
03426     float ws[dx + 1][dy + 1];
03427 #pragma omp parallel for default(shared) collapse(2)
03428     for (int ix = 0; ix <= dx; ix++)
03429         for (int iy = 0; iy < dy; iy++)
03430             ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03431                 * (1.0f - (float) iy / (float) dy);
03432
03433     /* Copy data... */
03434 #pragma omp parallel for default(shared) collapse(3)
03435     for (int ix = 0; ix < met->nx; ix++)
03436         for (int iy = 0; iy < met->ny; iy++)
03437             for (int ip = 0; ip < met->np; ip++)
03438                 help[ip][ix][iy] = met->z[ix][iy][ip];
03439
03440     /* Horizontal smoothing... */
03441 #pragma omp parallel for default(shared) collapse(3)
03442     for (int ip = 0; ip < met->np; ip++)
03443         for (int ix = 0; ix < met->nx; ix++)
03444             for (int iy = 0; iy < met->ny; iy++) {
03445                 float res = 0, wsum = 0;
03446                 int iy0 = GSL_MAX(iy - dy + 1, 0);
03447                 int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03448                 for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03449                     int ix3 = ix2;
03450                     if (ix3 < 0)
03451                         ix3 += met->nx;
03452                     else if (ix3 >= met->nx)
03453                         ix3 -= met->nx;
03454                     for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03455                         if (isfinite(help[ip][ix3][iy2])) {
03456                             float w = ws[abs(ix - ix2)][abs(iy - iy2)];

```

```

03457         res += w * help[ip][ix3][iy2];
03458         wsum += w;
03459     }
03460 }
03461 if (wsum > 0)
03462     met->z[ix][iy][ip] = res / wsum;
03463 else
03464     met->z[ix][iy][ip] = GSL_NAN;
03465 }
03466 }
03467
03468 /*****
03469 void read_met_grid(
03470     char *filename,
03471     int ncid,
03472     ctl_t * ctl,
03473     met_t * met) {
03474
03475     char levname[LEN], tstr[10];
03476
03477     double rtime, r2;
03478
03479     int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;
03480
03481     size_t np;
03482
03483     /* Set timer... */
03484     SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03485     LOG(2, "Read meteo grid information...");
03486
03487     /* MPTRAC meteo files... */
03488     if (ctl->clams_met_data == 0) {
03489
03490         /* Get time from filename... */
03491         size_t len = strlen(filename);
03492         sprintf(tstr, "%.4s", &filename[len - 16]);
03493         year = atoi(tstr);
03494         sprintf(tstr, "%.2s", &filename[len - 11]);
03495         mon = atoi(tstr);
03496         sprintf(tstr, "%.2s", &filename[len - 8]);
03497         day = atoi(tstr);
03498         sprintf(tstr, "%.2s", &filename[len - 5]);
03499         hour = atoi(tstr);
03500         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03501
03502         /* Check time information from data file... */
03503         if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03504             NC(nc_get_var_double(ncid, varid, &rtime));
03505             if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
03506                 WARN("Time information in meteo file does not match filename!");
03507         } else
03508             WARN("Time information in meteo file is missing!");
03509     }
03510
03511     /* CLaMS meteo files... */
03512     else {
03513
03514         /* Read time from file... */
03515         NC_GET_DOUBLE("time", &rtime, 0);
03516
03517         /* Get time from filename (considering the century)... */
03518         if (rtime < 0)
03519             sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
03520         else
03521             sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
03522         year = atoi(tstr);
03523         sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
03524         mon = atoi(tstr);
03525         sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
03526         day = atoi(tstr);
03527         sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03528         hour = atoi(tstr);
03529         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03530     }
03531
03532     /* Check time... */
03533     if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03534         || day < 1 || day > 31 || hour < 0 || hour > 23)
03535         ERRMSG("Cannot read time from filename!");
03536     jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03537     LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03538         met->time, year2, mon2, day2, hour2, min2);
03539
03540     /* Get grid dimensions... */
03541     NC_INQ_DIM("lon", &met->nx, 2, EX);
03542     LOG(2, "Number of longitudes: %d", met->nx);

```

```

03544
03545 NC_INQ_DIM("lat", &met->ny, 2, EY);
03546 LOG(2, "Number of latitudes: %d", met->ny);
03547
03548 if (ctl->vert_coord_met == 0) {
03549     sprintf(levname, "lev");
03550 } else {
03551     sprintf(levname, "hybrid");
03552     printf("Meteorological data is in hybrid coordinates.");
03553 }
03554 NC_INQ_DIM(levname, &met->np, 1, EP);
03555 if (met->np == 1) {
03556     int dimid;
03557     sprintf(levname, "lev_2");
03558     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03559         sprintf(levname, "plev");
03560         nc_inq_dimid(ncid, levname, &dimid);
03561     }
03562     NC(nc_inq_dimlen(ncid, dimid, &np));
03563     met->np = (int) np;
03564 }
03565 LOG(2, "Number of levels: %d", met->np);
03566 if (met->np < 2 || met->np > EP)
03567     ERRMSG("Number of levels out of range!");
03568
03569 /* Read longitudes and latitudes... */
03570 NC_GET_DOUBLE("lon", met->lon, 1);
03571 LOG(2, "Longitudes: %g, %g ... %g deg",
03572     met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03573 NC_GET_DOUBLE("lat", met->lat, 1);
03574 LOG(2, "Latitudes: %g, %g ... %g deg",
03575     met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03576
03577 /* Read pressure levels... */
03578 if (ctl->met_np <= 0) {
03579     NC_GET_DOUBLE(levname, met->p, 1);
03580     for (int ip = 0; ip < met->np; ip++)
03581         met->p[ip] /= 100.;
03582     LOG(2, "Altitude levels: %g, %g ... %g km",
03583         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03584     LOG(2, "Pressure levels: %g, %g ... %g hPa",
03585         met->p[0], met->p[1], met->p[met->np - 1]);
03586 }
03587 }
03588
03589 /*****
03590
03591 void read_met_levels(
03592     int ncid,
03593     ctl_t * ctl,
03594     met_t * met) {
03595
03596     /* Set timer... */
03597     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03598     LOG(2, "Read level data...");
03599
03600     /* MPTRAC meteo data... */
03601     if (ctl->clams_met_data == 0) {
03602
03603         /* Read meteo data... */
03604         if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
03605             ERRMSG("Cannot read temperature!");
03606         if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03607             ERRMSG("Cannot read zonal wind!");
03608         if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03609             ERRMSG("Cannot read meridional wind!");
03610         if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
03611             WARN("Cannot read vertical velocity!");
03612         if (!read_met_nc_3d
03613             (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03614             WARN("Cannot read specific humidity!");
03615         if (!read_met_nc_3d
03616             (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03617             WARN("Cannot read ozone data!");
03618         if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03619             if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03620                 WARN("Cannot read cloud liquid water content!");
03621             if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03622                 WARN("Cannot read cloud ice water content!");
03623         }
03624         if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03625             if (!read_met_nc_3d
03626                 (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03627                 ctl->met_cloud == 2))
03628                 WARN("Cannot read cloud rain water content!");
03629             if (!read_met_nc_3d
03630                 (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,

```

```

03631         ctl->met_cloud == 2))
03632         WARN("Cannot read cloud snow water content!");
03633     }
03634
03635     /* Transfer from model levels to pressure levels... */
03636     if (ctl->met_np > 0) {
03637
03638         /* Read pressure on model levels... */
03639         if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->pl, 0.01f, 1))
03640             ERRMSG("Cannot read pressure on model levels!");
03641
03642         /* Vertical interpolation from model to pressure levels... */
03643         read_met_ml2pl(ctl, met, met->t);
03644         read_met_ml2pl(ctl, met, met->u);
03645         read_met_ml2pl(ctl, met, met->v);
03646         read_met_ml2pl(ctl, met, met->w);
03647         read_met_ml2pl(ctl, met, met->h2o);
03648         read_met_ml2pl(ctl, met, met->o3);
03649         read_met_ml2pl(ctl, met, met->lwc);
03650         read_met_ml2pl(ctl, met, met->iwc);
03651
03652         /* Set new pressure levels... */
03653         met->np = ctl->met_np;
03654         for (int ip = 0; ip < met->np; ip++)
03655             met->p[ip] = ctl->met_p[ip];
03656     }
03657 }
03658
03659 /* CLaMS meteo data... */
03660 else if (ctl->clams_meteo_data == 1) {
03661
03662     /* Read meteorological data... */
03663     if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
03664         ERRMSG("Cannot read temperature!");
03665     if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03666         ERRMSG("Cannot read zonal wind!");
03667     if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03668         ERRMSG("Cannot read meridional wind!");
03669     if (!read_met_nc_3d(ncid, "w", "OMEGA", ctl, met, met->w, 0.01f, 1))
03670         WARN("Cannot read vertical velocity!");
03671     if (!read_met_nc_3d(ncid, "ZETA", "zeta", ctl, met, met->zeta, 1.0, 1))
03672         WARN("Cannot read ZETA in meteo data!");
03673     if (ctl->vert_vel == 1) {
03674         if (!read_met_nc_3d
03675             (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
03676              0.00001157407f, 1)) {
03677             if (!read_met_nc_3d
03678                 (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
03679                  0.00001157407f, 1)) {
03680                 WARN("Cannot read vertical velocity!");
03681             }
03682         }
03683     }
03684
03685     if (!read_met_nc_3d
03686         (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03687         WARN("Cannot read specific humidity!");
03688     if (!read_met_nc_3d
03689         (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03690         WARN("Cannot read ozone data!");
03691     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03692         if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03693             WARN("Cannot read cloud liquid water content!");
03694         if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03695             WARN("Cannot read cloud ice water content!");
03696     }
03697     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03698         if (!read_met_nc_3d
03699             (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03700              ctl->met_cloud == 2))
03701             WARN("Cannot read cloud rain water content!");
03702         if (!read_met_nc_3d
03703             (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03704              ctl->met_cloud == 2))
03705             WARN("Cannot read cloud snow water content!");
03706     }
03707
03708     /* Transfer from model levels to pressure levels... */
03709     if (ctl->met_np > 0) {
03710
03711         /* Read pressure on model levels... */
03712         if (!read_met_nc_3d(ncid, "pl", "PRESS", ctl, met, met->pl, 1.0, 1))
03713             ERRMSG("Cannot read pressure on model levels!");
03714
03715         /* Vertical interpolation from model to pressure levels... */
03716         read_met_ml2pl(ctl, met, met->t);
03717         read_met_ml2pl(ctl, met, met->u);

```



```

03718     read_met_ml2pl(ctl, met, met->v);
03719     read_met_ml2pl(ctl, met, met->w);
03720     read_met_ml2pl(ctl, met, met->h2o);
03721     read_met_ml2pl(ctl, met, met->o3);
03722     read_met_ml2pl(ctl, met, met->lwc);
03723     read_met_ml2pl(ctl, met, met->iwc);
03724     if (ctl->vert_vel == 1) {
03725         read_met_ml2pl(ctl, met, met->zeta);
03726         read_met_ml2pl(ctl, met, met->zeta_dot);
03727     }
03728
03729     /* Set new pressure levels... */
03730     met->np = ctl->met_np;
03731     for (int ip = 0; ip < met->np; ip++)
03732         met->p[ip] = ctl->met_p[ip];
03733
03734     /* Create a pressure field... */
03735     for (int i = 0; i < met->nx; i++)
03736         for (int j = 0; j < met->ny; j++)
03737             for (int k = 0; k < met->np; k++) {
03738                 met->patp[i][j][k] = (float) met->p[k];
03739             }
03740 }
03741 } else
03742     ERRMSG("Meteo data format unknown!");
03743
03744 /* Check ordering of pressure levels... */
03745 for (int ip = 1; ip < met->np; ip++)
03746     if (met->p[ip - 1] < met->p[ip])
03747         ERRMSG("Pressure levels must be descending!");
03748 }
03749
03750 /*****
03751 void read_met_ml2pl(
03752     ctl_t * ctl,
03753     met_t * met,
03754     float var[EX][EY][EP]) {
03755
03756     double aux[EP], p[EP];
03757
03758     /* Set timer... */
03759     SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03760     LOG(2, "Interpolate meteo data to pressure levels...");
03761
03762     /* Loop over columns... */
03763     #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03764     for (int ix = 0; ix < met->nx; ix++)
03765         for (int iy = 0; iy < met->ny; iy++) {
03766
03767             /* Copy pressure profile... */
03768             for (int ip = 0; ip < met->np; ip++)
03769                 p[ip] = met->pl[ix][iy][ip];
03770
03771             /* Interpolate... */
03772             for (int ip = 0; ip < ctl->met_np; ip++) {
03773                 double pt = ctl->met_p[ip];
03774                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03775                     pt = p[0];
03776                 else if ((pt > p[met->np - 1] && p[1] > p[0])
03777                     || (pt < p[met->np - 1] && p[1] < p[0]))
03778                     pt = p[met->np - 1];
03779                 int ip2 = locate_irr(p, met->np, pt);
03780                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03781                     p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03782             }
03783
03784             /* Copy data... */
03785             for (int ip = 0; ip < ctl->met_np; ip++)
03786                 var[ix][iy][ip] = (float) aux[ip];
03787         }
03788 }
03789
03790
03791 /*****
03792 int read_met_nc_2d(
03793     int ncid,
03794     char *varname,
03795     char *varname2,
03796     ctl_t * ctl,
03797     met_t * met,
03798     float dest[EX][EY],
03799     float scl,
03800     int init) {
03801
03802     char varsel[LEN];
03803
03804

```

```

03805     float offset, scalfac;
03806
03807     int varid;
03808
03809     /* Check if variable exists... */
03810     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03811         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03812             WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
03813             return 0;
03814         } else {
03815             sprintf(varsel, "%s", varname2);
03816         } else
03817             sprintf(varsel, "%s", varname);
03818
03819     /* Read packed data... */
03820     if (ctl->met_nc_scale
03821         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03822         && nc_get_att_float(ncid, varid, "scale_factor",
03823                             &scalfac) == NC_NOERR) {
03824
03825         /* Allocate... */
03826         short *help;
03827         ALLOC(help, short,
03828              EX * EY * EP);
03829
03830         /* Read fill value and missing value... */
03831         short fillval, missval;
03832         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03833             fillval = 0;
03834         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03835             missval = 0;
03836
03837         /* Write info... */
03838         LOG(2, "Read 2-D variable: %s"
03839            " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03840            varsel, fillval, missval, scalfac, offset);
03841
03842         /* Read data... */
03843         NC(nc_get_var_short(ncid, varid, help));
03844
03845         /* Copy and check data... */
03846         #pragma omp parallel for default(shared) num_threads(12)
03847         for (int ix = 0; ix < met->nx; ix++)
03848             for (int iy = 0; iy < met->ny; iy++) {
03849                 if (init)
03850                     dest[ix][iy] = 0;
03851                 short aux = help[ARRAY_2D(iy, ix, met->nx)];
03852                 if ((fillval == 0 || aux != fillval)
03853                     && (missval == 0 || aux != missval)
03854                     && fabsf(aux * scalfac + offset) < 1e14f)
03855                     dest[ix][iy] += scl * (aux * scalfac + offset);
03856                 else
03857                     dest[ix][iy] = GSL_NAN;
03858             }
03859
03860         /* Free... */
03861         free(help);
03862     }
03863
03864     /* Unpacked data... */
03865     else {
03866
03867         /* Allocate... */
03868         float *help;
03869         ALLOC(help, float,
03870              EX * EY);
03871
03872         /* Read fill value and missing value... */
03873         float fillval, missval;
03874         if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03875             fillval = 0;
03876         if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03877             missval = 0;
03878
03879         /* Write info... */
03880         LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03881            varsel, fillval, missval);
03882
03883         /* Read data... */
03884         NC(nc_get_var_float(ncid, varid, help));
03885
03886         /* Copy and check data... */
03887         #pragma omp parallel for default(shared) num_threads(12)
03888         for (int ix = 0; ix < met->nx; ix++)
03889             for (int iy = 0; iy < met->ny; iy++) {
03890                 if (init)
03891                     dest[ix][iy] = 0;

```

```

03892         float aux = help[ARRAY_2D(iy, ix, met->nx)];
03893         if ((fillval == 0 || aux != fillval)
03894             && (missval == 0 || aux != missval)
03895             && fabsf(aux) < 1e14f)
03896             dest[ix][iy] += scl * aux;
03897         else
03898             dest[ix][iy] = GSL_NAN;
03899     }
03900
03901     /* Free... */
03902     free(help);
03903 }
03904
03905 /* Return... */
03906 return 1;
03907 }
03908
03909 /*****
03910
03911 int read_met_nc_3d(
03912     int ncid,
03913     char *varname,
03914     char *varname2,
03915     ctl_t *ctl,
03916     met_t *met,
03917     float dest[EX][EY][EP],
03918     float scl,
03919     int init) {
03920
03921     char varsel[LEN];
03922
03923     float offset, scalfac;
03924
03925     int varid;
03926
03927     /* Check if variable exists... */
03928     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03929         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03930             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03931             return 0;
03932         } else {
03933             sprintf(varsel, "%s", varname2);
03934         } else
03935             sprintf(varsel, "%s", varname);
03936
03937     /* Read packed data... */
03938     if (ctl->met_nc_scale
03939         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03940         && nc_get_att_float(ncid, varid, "scale_factor",
03941                             &scalfac) == NC_NOERR) {
03942
03943         /* Allocate... */
03944         short *help;
03945         ALLOC(help, short,
03946              EX * EY * EP);
03947
03948         /* Read fill value and missing value... */
03949         short fillval, missval;
03950         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03951             fillval = 0;
03952         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03953             missval = 0;
03954
03955         /* Write info... */
03956         LOG(2, "Read 3-D variable: %s "
03957              "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03958              varsel, fillval, missval, scalfac, offset);
03959
03960         /* Read data... */
03961         NC(nc_get_var_short(ncid, varid, help));
03962
03963         /* Copy and check data... */
03964 #pragma omp parallel for default(shared) num_threads(12)
03965         for (int ix = 0; ix < met->nx; ix++)
03966             for (int iy = 0; iy < met->ny; iy++)
03967                 for (int ip = 0; ip < met->np; ip++) {
03968                     if (init)
03969                         dest[ix][iy][ip] = 0;
03970                     short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
03971                     if ((fillval == 0 || aux != fillval)
03972                         && (missval == 0 || aux != missval)
03973                         && fabsf(aux * scalfac + offset) < 1e14f)
03974                         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
03975                     else
03976                         dest[ix][iy][ip] = GSL_NAN;
03977                 }
03978

```

```

03979      /* Free... */
03980      free(help);
03981  }
03982
03983      /* Unpacked data... */
03984      else {
03985
03986          /* Allocate... */
03987          float *help;
03988          ALLOC(help, float,
03989                EX * EY * EP);
03990
03991          /* Read fill value and missing value... */
03992          float fillval, missval;
03993          if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03994              fillval = 0;
03995          if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03996              missval = 0;
03997
03998          /* Write info... */
03999          LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
04000              varsel, fillval, missval);
04001
04002          /* Read data... */
04003          NC(nc_get_var_float(ncid, varid, help));
04004
04005          /* Copy and check data... */
04006          #pragma omp parallel for default(shared) num_threads(12)
04007          for (int ix = 0; ix < met->nx; ix++)
04008              for (int iy = 0; iy < met->ny; iy++)
04009                  for (int ip = 0; ip < met->np; ip++) {
04010                      if (init)
04011                          dest[ix][iy][ip] = 0;
04012                      float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04013                      if ((fillval == 0 || aux != fillval)
04014                          && (missval == 0 || aux != missval)
04015                          && fabsf(aux) < 1e14f)
04016                          dest[ix][iy][ip] += scl * aux;
04017                      else
04018                          dest[ix][iy][ip] = GSL_NAN;
04019                  }
04020
04021          /* Free... */
04022          free(help);
04023      }
04024
04025      /* Return... */
04026      return 1;
04027  }
04028
04029  /*****
04030
04031  void read_met_pbl(
04032      met_t * met) {
04033
04034      /* Set timer... */
04035      SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
04036      LOG(2, "Calculate planetary boundary layer...");
04037
04038      /* Parameters used to estimate the height of the PBL
04039       (e.g., Vogelesang and Holtslag, 1996; Seidel et al., 2012)... */
04040      const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
04041
04042      /* Loop over grid points... */
04043      #pragma omp parallel for default(shared) collapse(2)
04044      for (int ix = 0; ix < met->nx; ix++)
04045          for (int iy = 0; iy < met->ny; iy++) {
04046
04047              /* Set bottom level of PBL... */
04048              double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
04049
04050              /* Find lowest level near the bottom... */
04051              int ip;
04052              for (ip = 1; ip < met->np; ip++)
04053                  if (met->p[ip] < pbl_bot)
04054                      break;
04055
04056              /* Get near surface data... */
04057              double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
04058                             met->p[ip], met->z[ix][iy][ip], pbl_bot);
04059              double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
04060                             met->p[ip], met->t[ix][iy][ip], pbl_bot);
04061              double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
04062                             met->p[ip], met->u[ix][iy][ip], pbl_bot);
04063              double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
04064                             met->p[ip], met->v[ix][iy][ip], pbl_bot);
04065              double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],

```

```

04066         met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
04067     double tvs = THETA_VIRT(pbl_bot, ts, h2os);
04068
04069     /* Init... */
04070     double rib_old = 0;
04071
04072     /* Loop over levels... */
04073     for (; ip < met->np; ip++) {
04074
04075         /* Get squared horizontal wind speed... */
04076         double vh2
04077             = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
04078         vh2 = GSL_MAX(vh2, SQR(umin));
04079
04080         /* Calculate bulk Richardson number... */
04081         double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
04082             * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
04083                 met->h2o[ix][iy][ip]) - tvs) / vh2;
04084
04085         /* Check for critical value... */
04086         if (rib >= rib_crit) {
04087             met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
04088                 rib, met->p[ip], rib_crit));
04089             if (met->pbl[ix][iy] > pbl_bot)
04090                 met->pbl[ix][iy] = (float) pbl_bot;
04091             break;
04092         }
04093
04094         /* Save Richardson number... */
04095         rib_old = rib;
04096     }
04097 }
04098 }
04099
04100 /*****
04101
04102 void read_met_periodic(
04103     met_t * met) {
04104
04105     /* Set timer... */
04106     SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
04107     LOG(2, "Apply periodic boundary conditions...");
04108
04109     /* Check longitudes... */
04110     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
04111         + met->lon[1] - met->lon[0] - 360) < 0.01))
04112         return;
04113
04114     /* Increase longitude counter... */
04115     if ((++met->nx) > EX)
04116         ERRMSG("Cannot create periodic boundary conditions!");
04117
04118     /* Set longitude... */
04119     met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
04120
04121     /* Loop over latitudes and pressure levels... */
04122     #pragma omp parallel for default(shared)
04123     for (int iy = 0; iy < met->ny; iy++) {
04124         met->ps[met->nx - 1][iy] = met->ps[0][iy];
04125         met->zs[met->nx - 1][iy] = met->zs[0][iy];
04126         met->ts[met->nx - 1][iy] = met->ts[0][iy];
04127         met->us[met->nx - 1][iy] = met->us[0][iy];
04128         met->vs[met->nx - 1][iy] = met->vs[0][iy];
04129         for (int ip = 0; ip < met->np; ip++) {
04130             met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
04131             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
04132             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
04133             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
04134             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
04135             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
04136             met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
04137             met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
04138         }
04139     }
04140 }
04141
04142 /*****
04143
04144 void read_met_pv(
04145     met_t * met) {
04146
04147     double pows[EP];
04148
04149     /* Set timer... */
04150     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
04151     LOG(2, "Calculate potential vorticity...");
04152

```

```

04153  /* Set powers... */
04154 #pragma omp parallel for default(shared)
04155 for (int ip = 0; ip < met->np; ip++)
04156     pows[ip] = pow(1000. / met->p[ip], 0.286);
04157
04158  /* Loop over grid points... */
04159 #pragma omp parallel for default(shared)
04160 for (int ix = 0; ix < met->nx; ix++) {
04161
04162     /* Set indices... */
04163     int ix0 = GSL_MAX(ix - 1, 0);
04164     int ix1 = GSL_MIN(ix + 1, met->nx - 1);
04165
04166     /* Loop over grid points... */
04167     for (int iy = 0; iy < met->ny; iy++) {
04168
04169         /* Set indices... */
04170         int iy0 = GSL_MAX(iy - 1, 0);
04171         int iy1 = GSL_MIN(iy + 1, met->ny - 1);
04172
04173         /* Set auxiliary variables... */
04174         double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
04175         double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
04176         double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
04177         double c0 = cos(met->lat[iy0] / 180. * M_PI);
04178         double c1 = cos(met->lat[iy1] / 180. * M_PI);
04179         double cr = cos(latr / 180. * M_PI);
04180         double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
04181
04182         /* Loop over grid points... */
04183         for (int ip = 0; ip < met->np; ip++) {
04184
04185             /* Get gradients in longitude... */
04186             double dtdx
04187                 = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
04188             double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
04189
04190             /* Get gradients in latitude... */
04191             double dtdy
04192                 = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
04193             double dudy
04194                 = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
04195
04196             /* Set indices... */
04197             int ip0 = GSL_MAX(ip - 1, 0);
04198             int ip1 = GSL_MIN(ip + 1, met->np - 1);
04199
04200             /* Get gradients in pressure... */
04201             double dtdp, dudp, dvdp;
04202             double dp0 = 100. * (met->p[ip] - met->p[ip0]);
04203             double dp1 = 100. * (met->p[ip1] - met->p[ip]);
04204             if (ip != ip0 && ip != ip1) {
04205                 double denom = dp0 * dp1 * (dp0 + dp1);
04206                 dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
04207                     - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
04208                     + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
04209                     / denom;
04210                 dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
04211                     - dp1 * dp1 * met->u[ix][iy][ip0]
04212                     + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
04213                     / denom;
04214                 dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
04215                     - dp1 * dp1 * met->v[ix][iy][ip0]
04216                     + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
04217                     / denom;
04218             } else {
04219                 double denom = dp0 + dp1;
04220                 dtdp =
04221                     (met->t[ix][iy][ip1] * pows[ip1] -
04222                     met->t[ix][iy][ip0] * pows[ip0]) / denom;
04223                 dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
04224                 dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
04225             }
04226
04227             /* Calculate PV... */
04228             met->pv[ix][iy][ip] = (float)
04229                 (1e6 * G0 *
04230                 (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
04231         }
04232     }
04233 }
04234
04235 /* Fix for polar regions... */
04236 #pragma omp parallel for default(shared)
04237 for (int ix = 0; ix < met->nx; ix++)
04238     for (int ip = 0; ip < met->np; ip++) {
04239         met->pv[ix][0][ip]

```

```

04240         = met->pv[ix][1][ip]
04241         = met->pv[ix][2][ip];
04242     met->pv[ix][met->ny - 1][ip]
04243     = met->pv[ix][met->ny - 2][ip]
04244     = met->pv[ix][met->ny - 3][ip];
04245 }
04246 }
04247
04248 /*****
04249
04250 void read_met_sample(
04251     ctl_t * ctl,
04252     met_t * met) {
04253
04254     met_t *help;
04255
04256     /* Check parameters... */
04257     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
04258         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
04259         return;
04260
04261     /* Set timer... */
04262     SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
04263     LOG(2, "Downsampling of meteo data...");
04264
04265     /* Allocate... */
04266     ALLOC(help, met_t, 1);
04267
04268     /* Copy data... */
04269     help->nx = met->nx;
04270     help->ny = met->ny;
04271     help->np = met->np;
04272     memcpy(help->lon, met->lon, sizeof(met->lon));
04273     memcpy(help->lat, met->lat, sizeof(met->lat));
04274     memcpy(help->p, met->p, sizeof(met->p));
04275
04276     /* Smoothing... */
04277     for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
04278         for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
04279             for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
04280                 help->ps[ix][iy] = 0;
04281                 help->zs[ix][iy] = 0;
04282                 help->ts[ix][iy] = 0;
04283                 help->us[ix][iy] = 0;
04284                 help->vs[ix][iy] = 0;
04285                 help->t[ix][iy][ip] = 0;
04286                 help->u[ix][iy][ip] = 0;
04287                 help->v[ix][iy][ip] = 0;
04288                 help->w[ix][iy][ip] = 0;
04289                 help->h2o[ix][iy][ip] = 0;
04290                 help->o3[ix][iy][ip] = 0;
04291                 help->lwc[ix][iy][ip] = 0;
04292                 help->iwc[ix][iy][ip] = 0;
04293                 float wsum = 0;
04294                 for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
04295                     ix2++) {
04296                     int ix3 = ix2;
04297                     if (ix3 < 0)
04298                         ix3 += met->nx;
04299                     else if (ix3 >= met->nx)
04300                         ix3 -= met->nx;
04301
04302                     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
04303                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
04304                         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
04305                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
04306                             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
04307                                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
04308                                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
04309                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
04310                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
04311                             help->ts[ix][iy] += w * met->ts[ix3][iy2];
04312                             help->us[ix][iy] += w * met->us[ix3][iy2];
04313                             help->vs[ix][iy] += w * met->vs[ix3][iy2];
04314                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
04315                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
04316                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
04317                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
04318                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
04319                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
04320                             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
04321                             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
04322                             wsum += w;
04323                         }
04324                     }
04325                 help->ps[ix][iy] /= wsum;
04326                 help->zs[ix][iy] /= wsum;

```

```

04327         help->ts[ix][iy] /= wsum;
04328         help->us[ix][iy] /= wsum;
04329         help->vs[ix][iy] /= wsum;
04330         help->t[ix][iy][ip] /= wsum;
04331         help->u[ix][iy][ip] /= wsum;
04332         help->v[ix][iy][ip] /= wsum;
04333         help->w[ix][iy][ip] /= wsum;
04334         help->h2o[ix][iy][ip] /= wsum;
04335         help->o3[ix][iy][ip] /= wsum;
04336         help->lwc[ix][iy][ip] /= wsum;
04337         help->iwc[ix][iy][ip] /= wsum;
04338     }
04339 }
04340 }
04341
04342 /* Downsampling... */
04343 met->nx = 0;
04344 for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04345     met->lon[met->nx] = help->lon[ix];
04346     met->ny = 0;
04347     for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04348         met->lat[met->ny] = help->lat[iy];
04349         met->ps[met->nx][met->ny] = help->ps[ix][iy];
04350         met->zs[met->nx][met->ny] = help->zs[ix][iy];
04351         met->ts[met->nx][met->ny] = help->ts[ix][iy];
04352         met->us[met->nx][met->ny] = help->us[ix][iy];
04353         met->vs[met->nx][met->ny] = help->vs[ix][iy];
04354         met->np = 0;
04355         for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04356             met->p[met->nx][met->ny][met->np] = help->p[ix][iy][ip];
04357             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04358             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04359             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04360             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04361             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04362             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04363             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
04364             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04365             met->np++;
04366         }
04367         met->ny++;
04368     }
04369     met->nx++;
04370 }
04371
04372 /* Free... */
04373 free(help);
04374 }
04375
04376 /*****
04377 void read_met_surface(
04378     int ncid,
04379     met_t * met,
04380     ctl_t * ctl) {
04381
04382     /* Set timer... */
04383     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04384     LOG(2, "Read surface data...");
04385
04386     /* MPTRAC meteo data... */
04387     if (ctl->clams_met_data == 0) {
04388
04389         /* Read surface pressure... */
04390         if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
04391             if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04392                 WARN("Cannot not read surface pressure data (use lowest level!)");
04393                 for (int ix = 0; ix < met->nx; ix++)
04394                     for (int iy = 0; iy < met->ny; iy++)
04395                         met->ps[ix][iy] = (float) met->p[0];
04396             }
04397         } else
04398             for (int ix = 0; ix < met->nx; ix++)
04399                 for (int iy = 0; iy < met->ny; iy++)
04400                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04401
04402         /* Read geopotential height at the surface... */
04403         if (!read_met_nc_2d(ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
04404             if (!read_met_nc_2d(ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
04405                 WARN("Cannot read surface geopotential height!");
04406
04407         /* Read temperature at the surface... */
04408         if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
04409             WARN("Cannot read surface temperature!");
04410     }
04411 }
04412
04413

```



```

04414      /* Read zonal wind at the surface... */
04415      if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
04416          WARN("Cannot read surface zonal wind!");
04417
04418      /* Read meridional wind at the surface... */
04419      if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
04420          WARN("Cannot read surface meridional wind!");
04421  }
04422
04423      /* CLaMS meteo data... */
04424      else {
04425
04426          /* Read surface pressure... */
04427          if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04428              WARN("Cannot not read surface pressure data (use lowest level!)");
04429              for (int ix = 0; ix < met->nx; ix++)
04430                  for (int iy = 0; iy < met->ny; iy++)
04431                      met->ps[ix][iy] = (float) met->p[0];
04432          }
04433
04434          /* Read geopotential height at the surface
04435             (use lowermost level of 3-D data field)... */
04436          float *help;
04437          ALLOC(help, float,
04438                EX * EY * EP);
04439          memcpy(help, met->pl, sizeof(met->pl));
04440          if (!read_met_nc_3d
04441              (ncid, "gph", "GPH", ctl, met, met->pl, (float) (1e-3 / GO), 1)) {
04442              ERRMSG("Cannot read geopotential height!");
04443          } else
04444              for (int ix = 0; ix < met->nx; ix++)
04445                  for (int iy = 0; iy < met->ny; iy++)
04446                      met->zs[ix][iy] = met->pl[ix][iy][0];
04447          memcpy(met->pl, help, sizeof(met->pl));
04448          free(help);
04449
04450          /* Read temperature at the surface... */
04451          if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
04452              WARN("Cannot read surface temperature!");
04453
04454          /* Read zonal wind at the surface... */
04455          if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
04456              WARN("Cannot read surface zonal wind!");
04457
04458          /* Read meridional wind at the surface... */
04459          if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
04460              WARN("Cannot read surface meridional wind!");
04461      }
04462  }
04463
04464      /*****
04465
04466  void read_met_tropo(
04467      ctl_t * ctl,
04468      clim_t * clim,
04469      met_t * met) {
04470
04471      double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04472             th2[200], z[EP], z2[200];
04473
04474      /* Set timer... */
04475      SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04476      LOG(2, "Calculate tropopause...");
04477
04478      /* Get altitude and pressure profiles... */
04479      #pragma omp parallel for default(shared)
04480      for (int iz = 0; iz < met->np; iz++)
04481          z[iz] = Z(met->p[iz]);
04482      #pragma omp parallel for default(shared)
04483      for (int iz = 0; iz <= 190; iz++) {
04484          z2[iz] = 4.5 + 0.1 * iz;
04485          p2[iz] = P(z2[iz]);
04486      }
04487
04488      /* Do not calculate tropopause... */
04489      if (ctl->met_tropo == 0)
04490          #pragma omp parallel for default(shared) collapse(2)
04491          for (int ix = 0; ix < met->nx; ix++)
04492              for (int iy = 0; iy < met->ny; iy++)
04493                  met->pt[ix][iy] = GSL_NAN;
04494
04495      /* Use tropopause climatology... */
04496      else if (ctl->met_tropo == 1) {
04497          #pragma omp parallel for default(shared) collapse(2)
04498          for (int ix = 0; ix < met->nx; ix++)
04499              for (int iy = 0; iy < met->ny; iy++)
04500                  met->pt[ix][iy] = (float) clim_tropo(clim, met->time, met->lat[iy]);

```

```

04501     }
04502
04503     /* Use cold point... */
04504     else if (ctl->met_tropo == 2) {
04505
04506         /* Loop over grid points... */
04507         #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04508         for (int ix = 0; ix < met->nx; ix++)
04509             for (int iy = 0; iy < met->ny; iy++) {
04510
04511                 /* Interpolate temperature profile... */
04512                 for (int iz = 0; iz < met->np; iz++)
04513                     t[iz] = met->t[ix][iy][iz];
04514                 spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04515
04516                 /* Find minimum... */
04517                 int iz = (int) gsl_stats_min_index(t2, 1, 171);
04518                 if (iz > 0 && iz < 170)
04519                     met->pt[ix][iy] = (float) p2[iz];
04520                 else
04521                     met->pt[ix][iy] = GSL_NAN;
04522             }
04523     }
04524
04525     /* Use WMO definition... */
04526     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04527
04528         /* Loop over grid points... */
04529         #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04530         for (int ix = 0; ix < met->nx; ix++)
04531             for (int iy = 0; iy < met->ny; iy++) {
04532
04533                 /* Interpolate temperature profile... */
04534                 int iz;
04535                 for (iz = 0; iz < met->np; iz++)
04536                     t[iz] = met->t[ix][iy][iz];
04537                 spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04538
04539                 /* Find 1st tropopause... */
04540                 met->pt[ix][iy] = GSL_NAN;
04541                 for (iz = 0; iz <= 170; iz++) {
04542                     int found = 1;
04543                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04544                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04545                             ctl->met_tropo_lapse) {
04546                             found = 0;
04547                             break;
04548                         }
04549                     if (found) {
04550                         if (iz > 0 && iz < 170)
04551                             met->pt[ix][iy] = (float) p2[iz];
04552                         break;
04553                     }
04554                 }
04555
04556                 /* Find 2nd tropopause... */
04557                 if (ctl->met_tropo == 4) {
04558                     met->pt[ix][iy] = GSL_NAN;
04559                     for (; iz <= 170; iz++) {
04560                         int found = 1;
04561                         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04562                             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04563                                 ctl->met_tropo_lapse_sep) {
04564                                 found = 0;
04565                                 break;
04566                             }
04567                         if (found)
04568                             break;
04569                     }
04570                     for (; iz <= 170; iz++) {
04571                         int found = 1;
04572                         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04573                             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04574                                 ctl->met_tropo_lapse) {
04575                                 found = 0;
04576                                 break;
04577                             }
04578                         if (found) {
04579                             if (iz > 0 && iz < 170)
04580                                 met->pt[ix][iy] = (float) p2[iz];
04581                             break;
04582                         }
04583                     }
04584                 }
04585             }
04586     }
04587

```

```

04588  /* Use dynamical tropopause... */
04589  else if (ctl->met_tropo == 5) {
04590
04591      /* Loop over grid points... */
04592      #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04593      for (int ix = 0; ix < met->nx; ix++)
04594          for (int iy = 0; iy < met->ny; iy++) {
04595
04596              /* Interpolate potential vorticity profile... */
04597              for (int iz = 0; iz < met->np; iz++)
04598                  pv[iz] = met->pv[ix][iy][iz];
04599              spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04600
04601              /* Interpolate potential temperature profile... */
04602              for (int iz = 0; iz < met->np; iz++)
04603                  th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04604              spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04605
04606              /* Find dynamical tropopause... */
04607              met->pt[ix][iy] = GSL_NAN;
04608              for (int iz = 0; iz <= 170; iz++)
04609                  if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04610                      || th2[iz] >= ctl->met_tropo_theta) {
04611                      if (iz > 0 && iz < 170)
04612                          met->pt[ix][iy] = (float) p2[iz];
04613                      break;
04614                  }
04615          }
04616      }
04617
04618      else
04619          ERRMSG("Cannot calculate tropopause!");
04620
04621      /* Interpolate temperature, geopotential height, and water vapor vmr... */
04622      #pragma omp parallel for default(shared) collapse(2)
04623      for (int ix = 0; ix < met->nx; ix++)
04624          for (int iy = 0; iy < met->ny; iy++) {
04625              double h2ot, tt, zt;
04626              INTPOL_INIT;
04627              intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04628                                met->lat[iy], &tt, ci, cw, 1);
04629              intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04630                                met->lat[iy], &zt, ci, cw, 0);
04631              intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04632                                met->lat[iy], &h2ot, ci, cw, 0);
04633              met->tt[ix][iy] = (float) tt;
04634              met->zt[ix][iy] = (float) zt;
04635              met->h2ot[ix][iy] = (float) h2ot;
04636          }
04637      }
04638
04639      /*****
04640
04641 void read_obs(
04642     char *filename,
04643     double *rt,
04644     double *rz,
04645     double *rlon,
04646     double *rlat,
04647     double *robs,
04648     int *nobs) {
04649
04650     FILE *in;
04651
04652     char line[LEN];
04653
04654     /* Open observation data file... */
04655     LOG(1, "Read observation data: %s", filename);
04656     if (!(in = fopen(filename, "r")))
04657         ERRMSG("Cannot open file!");
04658
04659     /* Read observations... */
04660     while (fgets(line, LEN, in))
04661         if (sscanf(line, "%lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
04662                   &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
04663             if (++(*nobs) >= NOBS)
04664                 ERRMSG("Too many observations!");
04665
04666     /* Close observation data file... */
04667     fclose(in);
04668
04669     /* Check time... */
04670     for (int i = 1; i < *nobs; i++)
04671         if (rt[i] < rt[i - 1])
04672             ERRMSG("Time must be ascending!");
04673
04674     /* Write info... */

```

```

04675     int n = *nobs;
04676     double mini, maxi;
04677     LOG(2, "Number of observations: %d", *nobs);
04678     gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
04679     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04680     gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
04681     LOG(2, "Altitude range: %g ... %g km", mini, maxi);
04682     gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
04683     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04684     gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
04685     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04686     gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
04687     LOG(2, "Observation range: %g ... %g", mini, maxi);
04688 }
04689
04690 /*****
04691
04692 double scan_ctl(
04693     const char *filename,
04694     int argc,
04695     char *argv[],
04696     const char *varname,
04697     int arridx,
04698     const char *defvalue,
04699     char *value) {
04700
04701     FILE *in = NULL;
04702
04703     char fullname1[LEN], fullname2[LEN], rval[LEN];
04704
04705     int contain = 0, i;
04706
04707     /* Open file... */
04708     if (filename[strlen(filename) - 1] != '-')
04709         if (!(in = fopen(filename, "r")))
04710             ERRMSG("Cannot open file!");
04711
04712     /* Set full variable name... */
04713     if (arridx >= 0) {
04714         sprintf(fullname1, "%s[%d]", varname, arridx);
04715         sprintf(fullname2, "%s[*]", varname);
04716     } else {
04717         sprintf(fullname1, "%s", varname);
04718         sprintf(fullname2, "%s", varname);
04719     }
04720
04721     /* Read data... */
04722     if (in != NULL) {
04723         char dummy[LEN], line[LEN], rvarname[LEN];
04724         while (fgets(line, LEN, in)) {
04725             if (sscanf(line, "%4999s %4999s", rvarname, dummy, rval) == 3)
04726                 if (strcasecmp(rvarname, fullname1) == 0 ||
04727                     strcasecmp(rvarname, fullname2) == 0) {
04728                     contain = 1;
04729                     break;
04730                 }
04731         }
04732     }
04733     for (i = 1; i < argc - 1; i++)
04734         if (strcasecmp(argv[i], fullname1) == 0 ||
04735             strcasecmp(argv[i], fullname2) == 0) {
04736             sprintf(rval, "%s", argv[i + 1]);
04737             contain = 1;
04738             break;
04739         }
04740
04741     /* Close file... */
04742     if (in != NULL)
04743         fclose(in);
04744
04745     /* Check for missing variables... */
04746     if (!contain) {
04747         if (strlen(defvalue) > 0)
04748             sprintf(rval, "%s", defvalue);
04749         else
04750             ERRMSG("Missing variable %s!\n", fullname1);
04751     }
04752
04753     /* Write info... */
04754     LOG(1, "%s = %s", fullname1, rval);
04755
04756     /* Return values... */
04757     if (value != NULL)
04758         sprintf(value, "%s", rval);
04759     return atof(rval);
04760 }
04761

```

```

04762 /*****
04763
04764 double sedi(
04765     double p,
04766     double T,
04767     double rp,
04768     double rhop) {
04769
04770     /* Convert particle radius from microns to m... */
04771     rp *= 1e-6;
04772
04773     /* Density of dry air [kg / m^3]... */
04774     double rho = RHO(p, T);
04775
04776     /* Dynamic viscosity of air [kg / (m s)]... */
04777     double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04778
04779     /* Thermal velocity of an air molecule [m / s]... */
04780     double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04781
04782     /* Mean free path of an air molecule [m]... */
04783     double lambda = 2. * eta / (rho * v);
04784
04785     /* Knudsen number for air (dimensionless)... */
04786     double K = lambda / rp;
04787
04788     /* Cunningham slip-flow correction (dimensionless)... */
04789     double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04790
04791     /* Sedimentation velocity [m / s]... */
04792     return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
04793 }
04794
04795 /*****
04796
04797 void spline(
04798     double *x,
04799     double *y,
04800     int n,
04801     double *x2,
04802     double *y2,
04803     int n2,
04804     int method) {
04805
04806     /* Cubic spline interpolation... */
04807     if (method == 1) {
04808
04809         /* Allocate... */
04810         gsl_interp_accel *acc;
04811         gsl_spline *s;
04812         acc = gsl_interp_accel_alloc();
04813         s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04814
04815         /* Interpolate profile... */
04816         gsl_spline_init(s, x, y, (size_t) n);
04817         for (int i = 0; i < n2; i++)
04818             if (x2[i] <= x[0])
04819                 y2[i] = y[0];
04820             else if (x2[i] >= x[n - 1])
04821                 y2[i] = y[n - 1];
04822             else
04823                 y2[i] = gsl_spline_eval(s, x2[i], acc);
04824
04825         /* Free... */
04826         gsl_spline_free(s);
04827         gsl_interp_accel_free(acc);
04828     }
04829
04830     /* Linear interpolation... */
04831     else {
04832         for (int i = 0; i < n2; i++)
04833             if (x2[i] <= x[0])
04834                 y2[i] = y[0];
04835             else if (x2[i] >= x[n - 1])
04836                 y2[i] = y[n - 1];
04837             else {
04838                 int idx = locate_irr(x, n, x2[i]);
04839                 y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04840             }
04841     }
04842 }
04843
04844 /*****
04845
04846 float stddev(
04847     float *data,
04848     int n) {

```

```

04849
04850     if (n <= 0)
04851         return 0;
04852
04853     float mean = 0, var = 0;
04854
04855     for (int i = 0; i < n; ++i) {
04856         mean += data[i];
04857         var += SQR(data[i]);
04858     }
04859
04860     var = var / (float) n - SQR(mean / (float) n);
04861
04862     return (var > 0 ? sqrtf(var) : 0);
04863 }
04864
04865 /*****
04866
04867 double sza(
04868     double sec,
04869     double lon,
04870     double lat) {
04871
04872     double D, dec, e, g, GMST, h, L, LST, q, ra;
04873
04874     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
04875     D = sec / 86400 - 0.5;
04876
04877     /* Geocentric apparent ecliptic longitude [rad]... */
04878     g = (357.529 + 0.98560028 * D) * M_PI / 180;
04879     q = 280.459 + 0.98564736 * D;
04880     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
04881
04882     /* Mean obliquity of the ecliptic [rad]... */
04883     e = (23.439 - 0.00000036 * D) * M_PI / 180;
04884
04885     /* Declination [rad]... */
04886     dec = asin(sin(e) * sin(L));
04887
04888     /* Right ascension [rad]... */
04889     ra = atan2(cos(e) * sin(L), cos(L));
04890
04891     /* Greenwich Mean Sidereal Time [h]... */
04892     GMST = 18.697374558 + 24.06570982441908 * D;
04893
04894     /* Local Sidereal Time [h]... */
04895     LST = GMST + lon / 15;
04896
04897     /* Hour angle [rad]... */
04898     h = LST / 12 * M_PI - ra;
04899
04900     /* Convert latitude... */
04901     lat *= M_PI / 180;
04902
04903     /* Return solar zenith angle [rad]... */
04904     return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
04905 }
04906
04907 /*****
04908
04909 void time2jsec(
04910     int year,
04911     int mon,
04912     int day,
04913     int hour,
04914     int min,
04915     int sec,
04916     double remain,
04917     double *jsec) {
04918
04919     struct tm t0, t1;
04920
04921     t0.tm_year = 100;
04922     t0.tm_mon = 0;
04923     t0.tm_mday = 1;
04924     t0.tm_hour = 0;
04925     t0.tm_min = 0;
04926     t0.tm_sec = 0;
04927
04928     t1.tm_year = year - 1900;
04929     t1.tm_mon = mon - 1;
04930     t1.tm_mday = day;
04931     t1.tm_hour = hour;
04932     t1.tm_min = min;
04933     t1.tm_sec = sec;
04934
04935     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;

```

```

04936 }
04937
04938 /*****
04939
04940 void timer(
04941     const char *name,
04942     const char *group,
04943     int output) {
04944
04945     static char names[NTIMER][100], groups[NTIMER][100];
04946
04947     static double rt_name[NTIMER], rt_group[NTIMER],
04948         rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;
04949
04950     static int iname = -1, igroup = -1, nname, ngroup, ct_name[NTIMER];
04951
04952     /* Get time... */
04953     t1 = omp_get_wtime();
04954     dt = t1 - t0;
04955
04956     /* Add elapsed time to current timers... */
04957     if (iname >= 0) {
04958         rt_name[iname] += dt;
04959         rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
04960         rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
04961         ct_name[iname]++;
04962     }
04963     if (igroup >= 0)
04964         rt_group[igroup] += t1 - t0;
04965
04966     /* Report timers... */
04967     if (output) {
04968         for (int i = 0; i < nname; i++)
04969             LOG(1, "TIMER_%s = %.3f s      (min= %g s, mean= %g s,"
04970                 " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
04971                 rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
04972         for (int i = 0; i < ngroup; i++)
04973             LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
04974         double total = 0.0;
04975         for (int i = 0; i < nname; i++)
04976             total += rt_name[i];
04977         LOG(1, "TIMER_TOTAL = %.3f s", total);
04978     }
04979
04980     /* Identify IDs of next timer... */
04981     for (iname = 0; iname < nname; iname++)
04982         if (strcasecmp(name, names[iname]) == 0)
04983             break;
04984     for (igroup = 0; igroup < ngroup; igroup++)
04985         if (strcasecmp(group, groups[igroup]) == 0)
04986             break;
04987
04988     /* Check whether this is a new timer... */
04989     if (iname >= nname) {
04990         sprintf(names[iname], "%s", name);
04991         if ((++nname) > NTIMER)
04992             ERRMSG("Too many timers!");
04993     }
04994
04995     /* Check whether this is a new group... */
04996     if (igroup >= ngroup) {
04997         sprintf(groups[igroup], "%s", group);
04998         if ((++ngroup) > NTIMER)
04999             ERRMSG("Too many groups!");
05000     }
05001
05002     /* Save starting time... */
05003     t0 = t1;
05004 }
05005
05006 /*****
05007
05008 double tropo_weight(
05009     clim_t * clim,
05010     double t,
05011     double lat,
05012     double p) {
05013
05014     /* Get tropopause pressure... */
05015     double pt = clim_tropo(clim, t, lat);
05016
05017     /* Get pressure range... */
05018     double p1 = pt * 0.866877899;
05019     double p0 = pt / 0.866877899;
05020
05021     /* Get weighting factor... */
05022     if (p > p0)

```

```

05023     return 1;
05024 else if (p < p1)
05025     return 0;
05026 else
05027     return LIN(p0, 1.0, p1, 0.0, p);
05028 }
05029
05030 /*****
05031
05032 void write_atm(
05033     const char *filename,
05034     ctl_t * ctl,
05035     atm_t * atm,
05036     double t) {
05037
05038     /* Set timer... */
05039     SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
05040
05041     /* Write info... */
05042     LOG(1, "Write atmospheric data: %s", filename);
05043
05044     /* Write ASCII data... */
05045     if (ctl->atm_type == 0)
05046         write_atm_asc(filename, ctl, atm, t);
05047
05048     /* Write binary data... */
05049     else if (ctl->atm_type == 1)
05050         write_atm_bin(filename, ctl, atm);
05051
05052     /* Write netCDF data... */
05053     else if (ctl->atm_type == 2)
05054         write_atm_nc(filename, ctl, atm);
05055
05056     /* Write CLaMS data... */
05057     else if (ctl->atm_type == 3)
05058         write_atm_clams(ctl, atm, t);
05059
05060     /* Error... */
05061     else
05062         ERRMSG("Atmospheric data type not supported!");
05063
05064     /* Write info... */
05065     double mini, maxi;
05066     LOG(2, "Number of particles: %d", atm->np);
05067     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
05068     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
05069     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
05070     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
05071     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
05072     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
05073     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
05074     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
05075     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
05076     for (int iq = 0; iq < ctl->nq; iq++) {
05077         char msg[LEN];
05078         sprintf(msg, "Quantity %s range: %s ... %s %s",
05079             ctl->qnt_name[iq], ctl->qnt_format[iq],
05080             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
05081         gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
05082         LOG(2, msg, mini, maxi);
05083     }
05084 }
05085
05086 /*****
05087
05088 void write_atm_asc(
05089     const char *filename,
05090     ctl_t * ctl,
05091     atm_t * atm,
05092     double t) {
05093
05094     FILE *out;
05095
05096     /* Set time interval for output... */
05097     double t0 = t - 0.5 * ctl->dt_mod;
05098     double t1 = t + 0.5 * ctl->dt_mod;
05099
05100     /* Check if gnuplot output is requested... */
05101     if (ctl->atm_gpfile[0] != '-') {
05102
05103         /* Create gnuplot pipe... */
05104         if (!(out = popen("gnuplot", "w")))
05105             ERRMSG("Cannot create pipe to gnuplot!");
05106
05107         /* Set plot filename... */
05108         fprintf(out, "set out \"%s.png\"\n", filename);
05109

```



```

05110     /* Set time string... */
05111     double r;
05112     int year, mon, day, hour, min, sec;
05113     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05114     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\\n\"",
05115             year, mon, day, hour, min);
05116
05117     /* Dump gnuplot file to pipe... */
05118     FILE *in;
05119     if (!(in = fopen(ctl->atm_gpfile, "r")))
05120         ERRMSG("Cannot open file!");
05121     char line[LEN];
05122     while (fgets(line, LEN, in))
05123         fprintf(out, "%s", line);
05124     fclose(in);
05125 }
05126
05127 else {
05128
05129     /* Create file... */
05130     if (!(out = fopen(filename, "w")))
05131         ERRMSG("Cannot create file!");
05132 }
05133
05134 /* Write header... */
05135 fprintf(out,
05136         "# $1 = time [s]\\n"
05137         "# $2 = altitude [km]\\n"
05138         "# $3 = longitude [deg]\\n" "# $4 = latitude [deg]\\n");
05139 for (int iq = 0; iq < ctl->nq; iq++)
05140     fprintf(out, "# $%i = %s [%s]\\n", iq + 5, ctl->qnt_name[iq],
05141             ctl->qnt_unit[iq]);
05142 fprintf(out, "\\n");
05143
05144 /* Write data... */
05145 for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
05146
05147     /* Check time... */
05148     if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05149         continue;
05150
05151     /* Write output... */
05152     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
05153             atm->lon[ip], atm->lat[ip]);
05154     for (int iq = 0; iq < ctl->nq; iq++) {
05155         fprintf(out, " ");
05156         if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05157             fprintf(out, ctl->qnt_format[iq], GSL_NAN);
05158         else
05159             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05160     }
05161     fprintf(out, "\\n");
05162 }
05163
05164 /* Close file... */
05165 fclose(out);
05166 }
05167
05168 /*****
05169
05170 void write_atm_bin(
05171     const char *filename,
05172     ctl_t * ctl,
05173     atm_t * atm) {
05174
05175     FILE *out;
05176
05177     /* Create file... */
05178     if (!(out = fopen(filename, "w")))
05179         ERRMSG("Cannot create file!");
05180
05181     /* Write version of binary data... */
05182     int version = 100;
05183     FWRITE(&version, int,
05184           1,
05185           out);
05186
05187     /* Write data... */
05188     FWRITE(&atm->np, int,
05189           1,
05190           out);
05191     FWRITE(atm->time, double,
05192           (size_t) atm->np,
05193           out);
05194     FWRITE(atm->p, double,
05195           (size_t) atm->np,
05196           out);

```

```

05197 FWRITE(atm->lon, double,
05198         (size_t) atm->np,
05199         out);
05200 FWRITE(atm->lat, double,
05201         (size_t) atm->np,
05202         out);
05203 for (int iq = 0; iq < ctl->nq; iq++)
05204     FWRITE(atm->q[iq], double,
05205           (size_t) atm->np,
05206           out);
05207
05208 /* Write final flag... */
05209 int final = 999;
05210 FWRITE(&final, int,
05211       1,
05212       out);
05213
05214 /* Close file... */
05215 fclose(out);
05216 }
05217
05218 /*****
05219
05220 void write_atm_clams(
05221     ctl_t * ctl,
05222     atm_t * atm,
05223     double t) {
05224
05225     /* Global Counter... */
05226     static size_t out_cnt = 0;
05227
05228     char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";
05229
05230     double r, r_start, r_stop;
05231
05232     int year, mon, day, hour, min, sec;
05233     int year_start, mon_start, day_start, hour_start, min_start, sec_start;
05234     int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
05235     int ncid, varid, tid, cid, zid, dim_ids[2];
05236
05237     /* time, nparc */
05238     size_t start[2], count[2];
05239
05240     /* Determine start and stop times of calculation... */
05241     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05242     jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
05243             &min_start, &sec_start, &r_start);
05244     jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
05245             &min_stop, &sec_stop, &r_stop);
05246
05247     /* Set filename... */
05248     sprintf(filename_out,
05249            "./traj_fix_3d_%02d%02d%02d%02d%02d%02d.nc",
05250            year_start % 100, mon_start, day_start, hour_start,
05251            year_stop % 100, mon_stop, day_stop, hour_stop);
05252     printf("Write traj file: %s\n", filename_out);
05253
05254     /* Define hyperslap for the traj_file... */
05255     start[0] = out_cnt;
05256     start[1] = 0;
05257     count[0] = 1;
05258     count[1] = (size_t) atm->np;
05259
05260     /* Create the file at the first timestep... */
05261     if (out_cnt == 0) {
05262
05263         /* Create file... */
05264         nc_create(filename_out, NC_CLOBBER, &ncid);
05265
05266         /* Define dimensions... */
05267         NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
05268         NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05269         NC(nc_def_dim(ncid, "TMDT", 7, &cid));
05270         dim_ids[0] = tid;
05271         dim_ids[1] = pid;
05272
05273         /* Define variables and their attributes... */
05274         NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05275                 "seconds since 2000-01-01 00:00:00 UTC");
05276         NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
05277         NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
05278         NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
05279         NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
05280         for (int iq = 0; iq < ctl->nq; iq++)
05281             NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05282                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05283

```

```

05284      /* Define global attributes... */
05285      NC_PUT_ATT("exp_VERTCOOR_name", "zeta");
05286      NC_PUT_ATT("model", "MPTRAC");
05287
05288      /* End definitions... */
05289      NC(nc_enddef(ncid));
05290      NC(nc_close(ncid));
05291  }
05292
05293  /* Increment global counter to change hyperslap... */
05294  out_cnt++;
05295
05296  /* Open file... */
05297  NC(nc_open(filename_out, NC_WRITE, &ncid));
05298
05299  /* Write data... */
05300  NC_PUT_DOUBLE("time", atm->time, 1);
05301  NC_PUT_DOUBLE("LAT", atm->lat, 1);
05302  NC_PUT_DOUBLE("LON", atm->lon, 1);
05303  NC_PUT_DOUBLE("PRESS", atm->p, 1);
05304  if (ctl->vert_coord_ap == 1) {
05305      NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
05306  } else if (ctl->qnt_zeta >= 0) {
05307      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
05308  }
05309  for (int iq = 0; iq < ctl->nq; iq++)
05310      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);
05311
05312  /* Close file... */
05313  NC(nc_close(ncid));
05314
05315  /* At the last time step create the init_fix_YYYYMMDDHH file... */
05316  if ((year == year_stop) && (mon == mon_stop)
05317      && (day == day_stop) && (hour == hour_stop)) {
05318
05319      /* Set filename... */
05320      char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
05321      sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
05322              year_stop % 100, mon_stop, day_stop, hour_stop);
05323      printf("Write init file: %s\n", filename_init);
05324
05325      /* Create file... */
05326      nc_create(filename_init, NC_CLOBBER, &ncid);
05327
05328      /* Define dimensions... */
05329      NC(nc_def_dim(ncid, "time", 1, &tid));
05330      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05331      dim_ids[0] = tid;
05332      dim_ids[1] = pid;
05333
05334      /* Define variables and their attributes... */
05335      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05336               "seconds since 2000-01-01 00:00:00 UTC");
05337      NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
05338      NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
05339      NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
05340      NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
05341      NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
05342      NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
05343      for (int iq = 0; iq < ctl->nq; iq++)
05344          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05345                   ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05346
05347      /* Define global attributes... */
05348      NC_PUT_ATT("exp_VERTCOOR_name", "zeta");
05349      NC_PUT_ATT("model", "MPTRAC");
05350
05351      /* End definitions... */
05352      NC(nc_enddef(ncid));
05353
05354      /* Write data... */
05355      NC_PUT_DOUBLE("time", atm->time, 0);
05356      NC_PUT_DOUBLE("LAT", atm->lat, 0);
05357      NC_PUT_DOUBLE("LON", atm->lon, 0);
05358      NC_PUT_DOUBLE("PRESS", atm->p, 0);
05359      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
05360      for (int iq = 0; iq < ctl->nq; iq++)
05361          NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05362
05363      /* Close file... */
05364      NC(nc_close(ncid));
05365  }
05366 }
05367
05368 /*****
05369
05370 void write_atm_nc(

```

```

05371     const char *filename,
05372     ctl_t * ctl,
05373     atm_t * atm) {
05374
05375     int ncid, obsid, varid;
05376
05377     size_t start[2], count[2];
05378
05379     /* Create file... */
05380     nc_create(filename, NC_CLOBBER, &ncid);
05381
05382     /* Define dimensions... */
05383     NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));
05384
05385     /* Define variables and their attributes... */
05386     NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
05387               "seconds since 2000-01-01 00:00:00 UTC");
05388     NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
05389     NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
05390     NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
05391     for (int iq = 0; iq < ctl->nq; iq++)
05392         NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
05393                   ctl->qnt_longname[iq], ctl->qnt_unit[iq]);
05394
05395     /* Define global attributes... */
05396     NC_PUT_ATT("featureType", "point");
05397
05398     /* End definitions... */
05399     NC(nc_enddef(ncid));
05400
05401     /* Write data... */
05402     NC_PUT_DOUBLE("time", atm->time, 0);
05403     NC_PUT_DOUBLE("press", atm->p, 0);
05404     NC_PUT_DOUBLE("lon", atm->lon, 0);
05405     NC_PUT_DOUBLE("lat", atm->lat, 0);
05406     for (int iq = 0; iq < ctl->nq; iq++)
05407         NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05408
05409     /* Close file... */
05410     NC(nc_close(ncid));
05411 }
05412
05413 /*****
05414 void write_csi(
05415     const char *filename,
05416     ctl_t * ctl,
05417     atm_t * atm,
05418     double t) {
05419
05420     static FILE *out;
05421
05422     static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
05423                 *dlon, *dlat, *dz, *x[NCSI], *y[NCSI];
05424
05425     static int *obscount, *ct, *cx, *cy, *cz, *ip, *ix, *iy, *iz, *n, *nobs;
05426
05427     /* Set timer... */
05428     SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
05429
05430     /* Init... */
05431     if (t == ctl->t_start) {
05432
05433         /* Check quantity index for mass... */
05434         if (ctl->qnt_m < 0)
05435             ERRMSG("Need quantity mass!");
05436
05437         /* Allocate... */
05438         ALLOC(area, double,
05439               ctl->csi_ny);
05440         ALLOC(rt, double,
05441               NOBS);
05442         ALLOC(rz, double,
05443               NOBS);
05444         ALLOC(rlon, double,
05445               NOBS);
05446         ALLOC(rlat, double,
05447               NOBS);
05448         ALLOC(robs, double,
05449               NOBS);
05450
05451         /* Read observation data... */
05452         read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);
05453
05454         /* Create new file... */
05455         LOG(1, "Write CSI data: %s", filename);
05456         if (!(out = fopen(filename, "w")))

```

```

05458     ERRMSG("Cannot create file!");
05459
05460     /* Write header... */
05461     fprintf(out,
05462         "# $1 = time [s]\n"
05463         "# $2 = number of hits (cx)\n"
05464         "# $3 = number of misses (cy)\n"
05465         "# $4 = number of false alarms (cz)\n"
05466         "# $5 = number of observations (cx + cy)\n"
05467         "# $6 = number of forecasts (cx + cz)\n"
05468         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
05469         "# $8 = probability of detection (POD) [%%]\n"
05470         "# $9 = false alarm rate (FAR) [%%]\n"
05471         "# $10 = critical success index (CSI) [%%]\n");
05472     fprintf(out,
05473         "# $11 = hits associated with random chance\n"
05474         "# $12 = equitable threat score (ETS) [%%]\n"
05475         "# $13 = Pearson linear correlation coefficient\n"
05476         "# $14 = Spearman rank-order correlation coefficient\n"
05477         "# $15 = column density mean error (F - O) [kg/m^2]\n"
05478         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
05479         "# $17 = column density mean absolute error [kg/m^2]\n"
05480         "# $18 = number of data points\n");
05481
05482     /* Set grid box size... */
05483     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
05484     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
05485     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
05486
05487     /* Set horizontal coordinates... */
05488     for (iy = 0; iy < ctl->csi_ny; iy++) {
05489         double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
05490         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
05491     }
05492 }
05493
05494     /* Set time interval... */
05495     double t0 = t - 0.5 * ctl->dt_mod;
05496     double t1 = t + 0.5 * ctl->dt_mod;
05497
05498     /* Allocate... */
05499     ALLOC(modmean, double,
05500         ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05501     ALLOC(obsmean, double,
05502         ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05503     ALLOC(obscount, int,
05504         ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05505
05506     /* Loop over observations... */
05507     for (int i = 0; i < nobs; i++) {
05508
05509         /* Check time... */
05510         if (rt[i] < t0)
05511             continue;
05512         else if (rt[i] >= t1)
05513             break;
05514
05515         /* Check observation data... */
05516         if (!isfinite(robs[i]))
05517             continue;
05518
05519         /* Calculate indices... */
05520         ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
05521         iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
05522         iz = (int) ((rz[i] - ctl->csi_z0) / dz);
05523
05524         /* Check indices... */
05525         if (ix < 0 || ix >= ctl->csi_nx ||
05526             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05527             continue;
05528
05529         /* Get mean observation index... */
05530         int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05531         obsmean[idx] += robs[i];
05532         obscount[idx]++;
05533     }
05534
05535     /* Analyze model data... */
05536     for (ip = 0; ip < atm->np; ip++) {
05537
05538         /* Check time... */
05539         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05540             continue;
05541
05542         /* Get indices... */
05543         ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
05544         iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);

```

```

05545     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
05546
05547     /* Check indices... */
05548     if (ix < 0 || ix >= ctl->csi_nx ||
05549         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05550         continue;
05551
05552     /* Get total mass in grid cell... */
05553     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05554     modmean[idx] += atm->q[ctl->qnt_m][ip];
05555 }
05556
05557 /* Analyze all grid cells... */
05558 for (ix = 0; ix < ctl->csi_nx; ix++)
05559     for (iy = 0; iy < ctl->csi_ny; iy++)
05560         for (iz = 0; iz < ctl->csi_nz; iz++) {
05561
05562             /* Calculate mean observation index... */
05563             int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05564             if (obscount[idx] > 0)
05565                 obsmean[idx] /= obscount[idx];
05566
05567             /* Calculate column density... */
05568             if (modmean[idx] > 0)
05569                 modmean[idx] /= (1e6 * area[iy]);
05570
05571             /* Calculate CSI... */
05572             if (obscount[idx] > 0) {
05573                 ct++;
05574                 if (obsmean[idx] >= ctl->csi_obsmin &&
05575                     modmean[idx] >= ctl->csi_modmin)
05576                     cx++;
05577                 else if (obsmean[idx] >= ctl->csi_obsmin &&
05578                     modmean[idx] < ctl->csi_modmin)
05579                     cy++;
05580                 else if (obsmean[idx] < ctl->csi_obsmin &&
05581                     modmean[idx] >= ctl->csi_modmin)
05582                     cz++;
05583             }
05584
05585             /* Save data for other verification statistics... */
05586             if (obscount[idx] > 0
05587                 && (obsmean[idx] >= ctl->csi_obsmin
05588                     || modmean[idx] >= ctl->csi_modmin)) {
05589                 x[n] = modmean[idx];
05590                 y[n] = obsmean[idx];
05591                 if (++n > NCSI)
05592                     ERRMSG("Too many data points to calculate statistics!");
05593             }
05594         }
05595
05596 /* Write output... */
05597 if (fmod(t, ctl->csi_dt_out) == 0) {
05598
05599     /* Calculate verification statistics
05600      (https://www.cawcr.gov.au/projects/verification/) ... */
05601     static double work[2 * NCSI];
05602     int n_obs = cx + cy;
05603     int n_for = cx + cz;
05604     double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
05605     double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
05606     double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
05607     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
05608     double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
05609     double ets = (cx + cy + cz - cx_rd > 0) ?
05610         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
05611     double rho_p =
05612         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
05613     double rho_s =
05614         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
05615     for (int i = 0; i < n; i++)
05616         work[i] = x[i] - y[i];
05617     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
05618     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
05619         0.0) : GSL_NAN;
05620     double absdev =
05621         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
05622
05623     /* Write... */
05624     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %d\n",
05625         t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
05626         rho_p, rho_s, mean, rmse, absdev, n);
05627
05628     /* Set counters to zero... */
05629     n = ct = cx = cy = cz = 0;
05630 }
05631

```

```

05632  /* Free... */
05633  free(modmean);
05634  free(obsmean);
05635  free(obscount);
05636
05637  /* Finalize... */
05638  if (t == ctl->t_stop) {
05639
05640      /* Close output file... */
05641      fclose(out);
05642
05643      /* Free... */
05644      free(area);
05645      free(rt);
05646      free(rz);
05647      free(rlon);
05648      free(rlat);
05649      free(robs);
05650  }
05651 }
05652
05653 /*****
05654 void write_ens(
05655     const char *filename,
05656     ctl_t * ctl,
05657     atm_t * atm,
05658     double t) {
05659
05660     static FILE *out;
05661
05662     static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
05663         x[3], zm[NENS];
05664
05665     static int n[NENS];
05666
05667     /* Set timer... */
05668     SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
05669
05670     /* Check quantities... */
05671     if (ctl->qnt_ens < 0)
05672         ERRMSG("Missing ensemble IDs!");
05673
05674     /* Set time interval... */
05675     double t0 = t - 0.5 * ctl->dt_mod;
05676     double t1 = t + 0.5 * ctl->dt_mod;
05677
05678     /* Init... */
05679     for (int i = 0; i < NENS; i++) {
05680         for (int iq = 0; iq < ctl->nq; iq++)
05681             qm[iq][i] = qs[iq][i] = 0;
05682         xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
05683         n[i] = 0;
05684     }
05685
05686     /* Loop over air parcels... */
05687     for (int ip = 0; ip < atm->np; ip++) {
05688
05689         /* Check time... */
05690         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05691             continue;
05692
05693         /* Check ensemble ID... */
05694         if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
05695             ERRMSG("Ensemble ID is out of range!");
05696
05697         /* Get means... */
05698         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
05699         for (int iq = 0; iq < ctl->nq; iq++) {
05700             qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
05701             qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
05702         }
05703         xm[ctl->qnt_ens][0] += x[0];
05704         xm[ctl->qnt_ens][1] += x[1];
05705         xm[ctl->qnt_ens][2] += x[2];
05706         zm[ctl->qnt_ens] += Z(atm->p[ip]);
05707         n[ctl->qnt_ens]++;
05708     }
05709
05710     /* Create file... */
05711     LOG(1, "Write ensemble data: %s", filename);
05712     if (!(out = fopen(filename, "w")))
05713         ERRMSG("Cannot create file!");
05714
05715     /* Write header... */
05716     fprintf(out,
05717         "# $1 = time [s]\n"

```

```

05719         "# $2 = altitude [km]\n"
05720         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05721     for (int iq = 0; iq < ctl->nq; iq++)
05722         fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
05723             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05724     for (int iq = 0; iq < ctl->nq; iq++)
05725         fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
05726             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05727     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
05728
05729     /* Write data... */
05730     for (int i = 0; i < NENS; i++)
05731         if (n[i] > 0) {
05732             cart2geo(xm[i], &dummy, &lon, &lat);
05733             fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
05734             for (int iq = 0; iq < ctl->nq; iq++) {
05735                 fprintf(out, " ");
05736                 fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
05737             }
05738             for (int iq = 0; iq < ctl->nq; iq++) {
05739                 fprintf(out, " ");
05740                 double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
05741                 fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
05742             }
05743             fprintf(out, " %d\n", n[i]);
05744         }
05745
05746     /* Close file... */
05747     fclose(out);
05748 }
05749
05750 /*****
05751
05752 void write_grid(
05753     const char *filename,
05754     ctl_t * ctl,
05755     met_t * met0,
05756     met_t * met1,
05757     atm_t * atm,
05758     double t) {
05759
05760     double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;
05761
05762     int *ixs, *iys, *izs, *np;
05763
05764     /* Set timer... */
05765     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
05766
05767     /* Write info... */
05768     LOG(1, "Write grid data: %s", filename);
05769
05770     /* Allocate... */
05771     ALLOC(cd, double,
05772         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05773     ALLOC(mass, double,
05774         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05775     ALLOC(vmr_expl, double,
05776         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05777     ALLOC(vmr_impl, double,
05778         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05779     ALLOC(z, double,
05780         ctl->grid_nz);
05781     ALLOC(lon, double,
05782         ctl->grid_nx);
05783     ALLOC(lat, double,
05784         ctl->grid_ny);
05785     ALLOC(area, double,
05786         ctl->grid_ny);
05787     ALLOC(press, double,
05788         ctl->grid_nz);
05789     ALLOC(np, int,
05790         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05791     ALLOC(ixs, int,
05792         atm->np);
05793     ALLOC(iys, int,
05794         atm->np);
05795     ALLOC(izs, int,
05796         atm->np);
05797
05798     /* Set grid box size... */
05799     double dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
05800     double dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
05801     double dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
05802
05803     /* Set vertical coordinates... */
05804     #pragma omp parallel for default(shared)
05805     for (int iz = 0; iz < ctl->grid_nz; iz++) {

```



```

05806     z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
05807     press[iz] = P(z[iz]);
05808 }
05809
05810 /* Set horizontal coordinates... */
05811 for (int ix = 0; ix < ctl->grid_nx; ix++)
05812     lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
05813 #pragma omp parallel for default(shared)
05814 for (int iy = 0; iy < ctl->grid_ny; iy++) {
05815     lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
05816     area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05817         * cos(lat[iy] * M_PI / 180.);
05818 }
05819
05820 /* Set time interval for output... */
05821 double t0 = t - 0.5 * ctl->dt_mod;
05822 double t1 = t + 0.5 * ctl->dt_mod;
05823
05824 /* Get grid box indices... */
05825 #pragma omp parallel for default(shared)
05826 for (int ip = 0; ip < atm->np; ip++) {
05827     ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
05828     iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
05829     izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05830     if (atm->time[ip] < t0 || atm->time[ip] > t1
05831         || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05832         || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05833         || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05834         izs[ip] = -1;
05835 }
05836
05837 /* Average data... */
05838 for (int ip = 0; ip < atm->np; ip++)
05839     if (izs[ip] >= 0) {
05840         int idx =
05841             ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
05842         np[idx]++;
05843         if (ctl->qnt_m >= 0)
05844             mass[idx] += atm->q[ctl->qnt_m][ip];
05845         if (ctl->qnt_vmr >= 0)
05846             vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
05847     }
05848
05849 /* Get implicit vmr per particle... */
05850 if (ctl->qnt_vmrimpl >= 0)
05851     for (int ip = 0; ip < atm->np; ip++)
05852         if (izs[ip] >= 0) {
05853             double temp;
05854             INTPOL_INIT;
05855             intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
05856                 lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
05857             atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass
05858                 *
05859                 mass[ARRAY_3D
05860                     (ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz)]
05861                 / (RHO(press[izs[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
05862         }
05863
05864 /* Calculate column density and vmr... */
05865 #pragma omp parallel for default(shared)
05866 for (int ix = 0; ix < ctl->grid_nx; ix++)
05867     for (int iy = 0; iy < ctl->grid_ny; iy++)
05868         for (int iz = 0; iz < ctl->grid_nz; iz++) {
05869
05870             /* Get grid index... */
05871             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
05872
05873             /* Calculate column density... */
05874             cd[idx] = GSL_NAN;
05875             if (ctl->qnt_m >= 0)
05876                 cd[idx] = mass[idx] / (1e6 * area[iy]);
05877
05878             /* Calculate volume mixing ratio (implicit)... */
05879             vmr_impl[idx] = GSL_NAN;
05880             if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05881                 vmr_impl[idx] = 0;
05882                 if (mass[idx] > 0) {
05883
05884                     /* Get temperature... */
05885                     double temp;
05886                     INTPOL_INIT;
05887                     intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05888                         lon[ix], lat[iy], &temp, ci, cw, 1);
05889
05890                     /* Calculate volume mixing ratio... */
05891                     vmr_impl[idx] = MA / ctl->molmass * mass[idx]
05892                         / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);

```

```

05893     }
05894 }
05895
05896     /* Calculate volume mixing ratio (explicit)... */
05897     if (ctl->qnt_vmr >= 0 && np[idx] > 0)
05898         vmr_expl[idx] /= np[idx];
05899     else
05900         vmr_expl[idx] = GSL_NAN;
05901 }
05902
05903 /* Write ASCII data... */
05904 if (ctl->grid_type == 0)
05905     write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
05906                   t, z, lon, lat, area, dz, np);
05907
05908 /* Write netCDF data... */
05909 else if (ctl->grid_type == 1)
05910     write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
05911                  t, z, lon, lat, area, dz, np);
05912
05913 /* Error message... */
05914 else
05915     ERRMSG("Grid data format GRID_TYPE unknown!");
05916
05917 /* Free... */
05918 free(cd);
05919 free(mass);
05920 free(vmr_expl);
05921 free(vmr_impl);
05922 free(z);
05923 free(lon);
05924 free(lat);
05925 free(area);
05926 free(press);
05927 free(np);
05928 free(ixs);
05929 free(iys);
05930 free(izs);
05931 }
05932
05933 /*****
05934
05935 void write_grid_asc(
05936     const char *filename,
05937     ctl_t *ctl,
05938     double *cd,
05939     double *vmr_expl,
05940     double *vmr_impl,
05941     double t,
05942     double *z,
05943     double *lon,
05944     double *lat,
05945     double *area,
05946     double dz,
05947     int *np) {
05948
05949     FILE *in, *out;
05950
05951     char line[LEN];
05952
05953     /* Check if gnuplot output is requested... */
05954     if (ctl->grid_gpfile[0] != '-') {
05955
05956         /* Create gnuplot pipe... */
05957         if (!(out = popen("gnuplot", "w")))
05958             ERRMSG("Cannot create pipe to gnuplot!");
05959
05960         /* Set plot filename... */
05961         fprintf(out, "set out \"%s.png\"\n", filename);
05962
05963         /* Set time string... */
05964         double r;
05965         int year, mon, day, hour, min, sec;
05966         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05967         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05968                year, mon, day, hour, min);
05969
05970         /* Dump gnuplot file to pipe... */
05971         if (!(in = fopen(ctl->grid_gpfile, "r")))
05972             ERRMSG("Cannot open file!");
05973         while (fgets(line, LEN, in))
05974             fprintf(out, "%s", line);
05975         fclose(in);
05976     }
05977
05978     else {
05979

```

```

05980     /* Create file... */
05981     if (! (out = fopen(filename, "w")))
05982         ERRMSG("Cannot create file!");
05983 }
05984
05985 /* Write header... */
05986 fprintf(out,
05987     "# $1 = time [s]\n"
05988     "# $2 = altitude [km]\n"
05989     "# $3 = longitude [deg]\n"
05990     "# $4 = latitude [deg]\n"
05991     "# $5 = surface area [km^2]\n"
05992     "# $6 = layer depth [km]\n"
05993     "# $7 = number of particles [l]\n"
05994     "# $8 = column density (implicit) [kg/m^2]\n"
05995     "# $9 = volume mixing ratio (implicit) [ppv]\n"
05996     "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
05997
05998 /* Write data... */
05999 for (int ix = 0; ix < ctl->grid_nx; ix++) {
06000     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
06001         fprintf(out, "\n");
06002     for (int iy = 0; iy < ctl->grid_ny; iy++) {
06003         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
06004             fprintf(out, "\n");
06005         for (int iz = 0; iz < ctl->grid_nz; iz++) {
06006             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
06007             if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
06008                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
06009                     t, z[iz], lon[ix], lat[iy], area[iy], dz,
06010                     np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
06011         }
06012     }
06013 }
06014
06015 /* Close file... */
06016 fclose(out);
06017 }
06018
06019 /*****
06020
06021 void write_grid_nc(
06022     const char *filename,
06023     ctl_t *ctl,
06024     double *cd,
06025     double *vmr_expl,
06026     double *vmr_impl,
06027     double t,
06028     double *z,
06029     double *lon,
06030     double *lat,
06031     double *area,
06032     double dz,
06033     int *np) {
06034
06035     int ncid, dimid[10], varid;
06036
06037     size_t start[2], count[2];
06038
06039     /* Create file... */
06040     nc_create(filename, NC_CLOBBER, &ncid);
06041
06042     /* Define dimensions... */
06043     NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
06044     NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[1]));
06045     NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
06046     NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[3]));
06047     NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));
06048
06049     /* Define variables and their attributes... */
06050     NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
06051         "seconds since 2000-01-01 00:00:00 UTC");
06052     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[1], "longitude", "degrees_east");
06053     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
06054     NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[3], "altitude", "km");
06055     NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
06056     NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[4], "layer depth", "km");
06057     NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
06058     NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
06059         "volume mixing ratio (implicit)", "ppv");
06060     NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
06061         "volume mixing ratio (explicit)", "ppv");
06062     NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");
06063
06064     /* End definitions... */
06065     NC(nc_enddef(ncid));
06066

```

```

06067  /* Write data... */
06068  NC_PUT_DOUBLE("time", &t, 0);
06069  NC_PUT_DOUBLE("lon", lon, 0);
06070  NC_PUT_DOUBLE("lat", lat, 0);
06071  NC_PUT_DOUBLE("z", z, 0);
06072  NC_PUT_DOUBLE("area", area, 0);
06073  NC_PUT_DOUBLE("dz", &dz, 0);
06074  NC_PUT_DOUBLE("cd", cd, 0);
06075  NC_PUT_DOUBLE("vmr_impl", vmr_impl, 0);
06076  NC_PUT_DOUBLE("vmr_expl", vmr_expl, 0);
06077  NC_PUT_INT("np", np, 0);
06078
06079  /* Close file... */
06080  NC(nc_close(ncid));
06081 }
06082
06083 /*****
06084
06085 int write_met(
06086     char *filename,
06087     ctl_t * ctl,
06088     met_t * met) {
06089
06090     /* Set timer... */
06091     SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);
06092
06093     /* Write info... */
06094     LOG(1, "Write meteo data: %s", filename);
06095
06096     /* Check compression flags... */
06097     #ifndef ZFP
06098     if (ctl->met_type == 3)
06099         ERRMSG("zfp compression not supported!");
06100     #endif
06101     #ifndef ZSTD
06102     if (ctl->met_type == 4)
06103         ERRMSG("zstd compression not supported!");
06104     #endif
06105
06106     /* Write binary... */
06107     if (ctl->met_type >= 1 && ctl->met_type <= 4) {
06108
06109         /* Create file... */
06110         FILE *out;
06111         if (!(out = fopen(filename, "w")))
06112             ERRMSG("Cannot create file!");
06113
06114         /* Write type of binary data... */
06115         FWRITE(&ctl->met_type, int,
06116             1,
06117             out);
06118
06119         /* Write version of binary data... */
06120         int version = 100;
06121         FWRITE(&version, int,
06122             1,
06123             out);
06124
06125         /* Write grid data... */
06126         FWRITE(&met->time, double,
06127             1,
06128             out);
06129         FWRITE(&met->nx, int,
06130             1,
06131             out);
06132         FWRITE(&met->ny, int,
06133             1,
06134             out);
06135         FWRITE(&met->np, int,
06136             1,
06137             out);
06138         FWRITE(met->lon, double,
06139             (size_t) met->nx,
06140             out);
06141         FWRITE(met->lat, double,
06142             (size_t) met->ny,
06143             out);
06144         FWRITE(met->p, double,
06145             (size_t) met->np,
06146             out);
06147
06148         /* Write surface data... */
06149         write_met_bin_2d(out, met, met->ps, "PS");
06150         write_met_bin_2d(out, met, met->ts, "TS");
06151         write_met_bin_2d(out, met, met->zs, "ZS");
06152         write_met_bin_2d(out, met, met->us, "US");
06153         write_met_bin_2d(out, met, met->vs, "VS");

```

```

06154     write_met_bin_2d(out, met, met->pbl, "PBL");
06155     write_met_bin_2d(out, met, met->pt, "PT");
06156     write_met_bin_2d(out, met, met->tt, "TT");
06157     write_met_bin_2d(out, met, met->zt, "ZT");
06158     write_met_bin_2d(out, met, met->h2ot, "H2OT");
06159     write_met_bin_2d(out, met, met->pct, "PCT");
06160     write_met_bin_2d(out, met, met->pcb, "PCB");
06161     write_met_bin_2d(out, met, met->cl, "CL");
06162     write_met_bin_2d(out, met, met->plcl, "PLCL");
06163     write_met_bin_2d(out, met, met->plfc, "PLFC");
06164     write_met_bin_2d(out, met, met->pel, "PEL");
06165     write_met_bin_2d(out, met, met->cape, "CAPE");
06166     write_met_bin_2d(out, met, met->cin, "CIN");
06167
06168     /* Write level data... */
06169     write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
06170     write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
06171     write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
06172     write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
06173     write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
06174     write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
06175     write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
06176     write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
06177     write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
06178     write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);
06179
06180     /* Write final flag... */
06181     int final = 999;
06182     FWRITE(&final, int,
06183          1,
06184          out);
06185
06186     /* Close file... */
06187     fclose(out);
06188 }
06189
06190 return 0;
06191 }
06192
06193 /*****
06194
06195 void write_met_bin_2d(
06196     FILE * out,
06197     met_t * met,
06198     float var[EX][EY],
06199     char *varname) {
06200
06201     float *help;
06202
06203     /* Allocate... */
06204     ALLOC(help, float,
06205          EX * EY);
06206
06207     /* Copy data... */
06208     for (int ix = 0; ix < met->nx; ix++)
06209         for (int iy = 0; iy < met->ny; iy++)
06210             help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];
06211
06212     /* Write uncompressed data... */
06213     LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
06214     FWRITE(help, float,
06215          (size_t) (met->nx * met->ny),
06216          out);
06217
06218     /* Free... */
06219     free(help);
06220 }
06221
06222 /*****
06223
06224 void write_met_bin_3d(
06225     FILE * out,
06226     ctl_t * ctl,
06227     met_t * met,
06228     float var[EX][EY][EP],
06229     char *varname,
06230     int precision,
06231     double tolerance) {
06232
06233     float *help;
06234
06235     /* Allocate... */
06236     ALLOC(help, float,
06237          EX * EY * EP);
06238
06239     /* Copy data... */
06240     #pragma omp parallel for default(shared) collapse(2)

```

```

06241     for (int ix = 0; ix < met->nx; ix++)
06242         for (int iy = 0; iy < met->ny; iy++)
06243             for (int ip = 0; ip < met->np; ip++)
06244                 help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];
06245
06246     /* Write uncompressed data... */
06247     if (ctl->met_type == 1) {
06248         LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
06249         FWRITE(help, float,
06250             (size_t) (met->nx * met->ny * met->np),
06251             out);
06252     }
06253
06254     /* Write packed data... */
06255     else if (ctl->met_type == 2)
06256         compress_pack(varname, help, (size_t) (met->ny * met->nx),
06257             (size_t) met->np, 0, out);
06258
06259     /* Write zfp data... */
06260 #ifdef ZFP
06261     else if (ctl->met_type == 3)
06262         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
06263             tolerance, 0, out);
06264 #endif
06265
06266     /* Write zstd data... */
06267 #ifdef ZSTD
06268     else if (ctl->met_type == 4)
06269         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
06270             out);
06271 #endif
06272
06273     /* Unknown method... */
06274     else {
06275         ERRMSG("MET_TYPE not supported!");
06276         LOG(3, "%d %g", precision, tolerance);
06277     }
06278
06279     /* Free... */
06280     free(help);
06281 }
06282
06283 /*****
06284 void write_prof(
06285     const char *filename,
06286     ctl_t *ctl,
06287     met_t *met0,
06288     met_t *met1,
06289     atm_t *atm,
06290     double t) {
06291
06292     static FILE *out;
06293
06294     static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
06295         dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;
06296
06297     static int nobs, *obscount, ip, okay;
06298
06299     /* Set timer... */
06300     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
06301
06302     /* Init... */
06303     if (t == ctl->t_start) {
06304
06305         /* Check quantity index for mass... */
06306         if (ctl->qnt_m < 0)
06307             ERRMSG("Need quantity mass!");
06308
06309         /* Check molar mass... */
06310         if (ctl->molmass <= 0)
06311             ERRMSG("Specify molar mass!");
06312
06313         /* Allocate... */
06314         ALLOC(lon, double,
06315             ctl->prof_nx);
06316         ALLOC(lat, double,
06317             ctl->prof_ny);
06318         ALLOC(area, double,
06319             ctl->prof_ny);
06320         ALLOC(z, double,
06321             ctl->prof_nz);
06322         ALLOC(press, double,
06323             ctl->prof_nz);
06324         ALLOC(rt, double,
06325             NOBS);
06326         ALLOC(rz, double,

```

```

06328         NOBS);
06329     ALLOC(rlon, double,
06330         NOBS);
06331     ALLOC(rlat, double,
06332         NOBS);
06333     ALLOC(robs, double,
06334         NOBS);
06335
06336     /* Read observation data... */
06337     read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06338
06339     /* Create new output file... */
06340     LOG(1, "Write profile data: %s", filename);
06341     if (!out = fopen(filename, "w"))
06342         ERRMSG("Cannot create file!");
06343
06344     /* Write header... */
06345     fprintf(out,
06346         "# $1 = time [s]\n"
06347         "# $2 = altitude [km]\n"
06348         "# $3 = longitude [deg]\n"
06349         "# $4 = latitude [deg]\n"
06350         "# $5 = pressure [hPa]\n"
06351         "# $6 = temperature [K]\n"
06352         "# $7 = volume mixing ratio [ppv]\n"
06353         "# $8 = H2O volume mixing ratio [ppv]\n"
06354         "# $9 = O3 volume mixing ratio [ppv]\n"
06355         "# $10 = observed BT index [K]\n"
06356         "# $11 = number of observations\n");
06357
06358     /* Set grid box size... */
06359     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
06360     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
06361     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
06362
06363     /* Set vertical coordinates... */
06364     for (int iz = 0; iz < ctl->prof_nz; iz++) {
06365         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
06366         press[iz] = P(z[iz]);
06367     }
06368
06369     /* Set horizontal coordinates... */
06370     for (int ix = 0; ix < ctl->prof_nx; ix++)
06371         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
06372     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06373         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
06374         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
06375             * cos(lat[iy] * M_PI / 180.);
06376     }
06377 }
06378
06379 /* Set time interval... */
06380 double t0 = t - 0.5 * ctl->dt_mod;
06381 double t1 = t + 0.5 * ctl->dt_mod;
06382
06383 /* Allocate... */
06384 ALLOC(mass, double,
06385     ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
06386 ALLOC(obsmean, double,
06387     ctl->prof_nx * ctl->prof_ny);
06388 ALLOC(obscount, int,
06389     ctl->prof_nx * ctl->prof_ny);
06390
06391 /* Loop over observations... */
06392 for (int i = 0; i < nobs; i++) {
06393
06394     /* Check time... */
06395     if (rt[i] < t0)
06396         continue;
06397     else if (rt[i] >= t1)
06398         break;
06399
06400     /* Check observation data... */
06401     if (!isfinite(robs[i]))
06402         continue;
06403
06404     /* Calculate indices... */
06405     int ix = (int) ((rlon[i] - ctl->prof_lon0) / dlon);
06406     int iy = (int) ((rlat[i] - ctl->prof_lat0) / dlat);
06407
06408     /* Check indices... */
06409     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
06410         continue;
06411
06412     /* Get mean observation index... */
06413     int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
06414     obsmean[idx] += robs[i];

```

```

06415     obscount[idx]++;
06416 }
06417
06418 /* Analyze model data... */
06419 for (ip = 0; ip < atm->np; ip++) {
06420
06421     /* Check time... */
06422     if (atm->time[ip] < t0 || atm->time[ip] > t1)
06423         continue;
06424
06425     /* Get indices... */
06426     int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
06427     int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
06428     int iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
06429
06430     /* Check indices... */
06431     if (ix < 0 || ix >= ctl->prof_nx ||
06432         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
06433         continue;
06434
06435     /* Get total mass in grid cell... */
06436     int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06437     mass[idx] += atm->q[ctl->qnt_m][ip];
06438 }
06439
06440 /* Extract profiles... */
06441 for (int ix = 0; ix < ctl->prof_nx; ix++)
06442     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06443         int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
06444         if (obscount[idx2] > 0) {
06445
06446             /* Check profile... */
06447             okay = 0;
06448             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06449                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06450                 if (mass[idx3] > 0) {
06451                     okay = 1;
06452                     break;
06453                 }
06454             }
06455             if (!okay)
06456                 continue;
06457
06458             /* Write output... */
06459             fprintf(out, "\n");
06460
06461             /* Loop over altitudes... */
06462             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06463
06464                 /* Get temperature, water vapor, and ozone... */
06465                 INTPOL_INIT;
06466                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
06467                                     lon[ix], lat[iy], &temp, ci, cw, 1);
06468                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
06469                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
06470                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
06471                                     lon[ix], lat[iy], &o3, ci, cw, 0);
06472
06473                 /* Calculate volume mixing ratio... */
06474                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06475                 vmr = MA / ctl->molmass * mass[idx3]
06476                     / (RHO(press[iz], temp) * area[iy] * dz * 1e9);
06477
06478                 /* Write output... */
06479                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
06480                         t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
06481                         obsmean[idx2] / obscount[idx2], obscount[idx2]);
06482             }
06483         }
06484     }
06485
06486 /* Free... */
06487 free(mass);
06488 free(obsmean);
06489 free(obscount);
06490
06491 /* Finalize... */
06492 if (t == ctl->t_stop) {
06493
06494     /* Close output file... */
06495     fclose(out);
06496
06497     /* Free... */
06498     free(lon);
06499     free(lat);
06500     free(area);
06501     free(z);

```



```

06502     free(press);
06503     free(rt);
06504     free(rz);
06505     free(rlon);
06506     free(rlat);
06507     free(robs);
06508 }
06509 }
06510
06511 /*****
06512
06513 void write_sample(
06514     const char *filename,
06515     ctl_t * ctl,
06516     met_t * met0,
06517     met_t * met1,
06518     atm_t * atm,
06519     double t) {
06520
06521     static FILE *out;
06522
06523     static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;
06524
06525     static int nobs;
06526
06527     /* Set timer... */
06528     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
06529
06530     /* Init... */
06531     if (t == ctl->t_start) {
06532
06533         /* Allocate... */
06534         ALLOC(rt, double,
06535             NOBS);
06536         ALLOC(rz, double,
06537             NOBS);
06538         ALLOC(rlon, double,
06539             NOBS);
06540         ALLOC(rlat, double,
06541             NOBS);
06542         ALLOC(robs, double,
06543             NOBS);
06544
06545         /* Read observation data... */
06546         read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06547
06548         /* Create output file... */
06549         LOG(1, "Write sample data: %s", filename);
06550         if (!(out = fopen(filename, "w")))
06551             ERRMSG("Cannot create file!");
06552
06553         /* Write header... */
06554         fprintf(out,
06555             "# $1 = time [s]\n"
06556             "# $2 = altitude [km]\n"
06557             "# $3 = longitude [deg]\n"
06558             "# $4 = latitude [deg]\n"
06559             "# $5 = surface area [km^2]\n"
06560             "# $6 = layer depth [km]\n"
06561             "# $7 = number of particles [1]\n"
06562             "# $8 = column density [kg/m^2]\n"
06563             "# $9 = volume mixing ratio [ppv]\n"
06564             "# $10 = observed BT index [K]\n\n");
06565
06566         /* Set latitude range, squared radius, and area... */
06567         dlat = DY2DEG(ctl->sample_dx);
06568         rmax2 = SQR(ctl->sample_dx);
06569         area = M_PI * rmax2;
06570     }
06571
06572     /* Set time interval for output... */
06573     double t0 = t - 0.5 * ctl->dt_mod;
06574     double t1 = t + 0.5 * ctl->dt_mod;
06575
06576     /* Loop over observations... */
06577     for (int i = 0; i < nobs; i++) {
06578
06579         /* Check time... */
06580         if (rt[i] < t0)
06581             continue;
06582         else if (rt[i] >= t1)
06583             break;
06584
06585         /* Calculate Cartesian coordinates... */
06586         double x0[3];
06587         geo2cart(0, rlon[i], rlat[i], x0);
06588

```

```

06589      /* Set pressure range... */
06590      double rp = P(rz[i]);
06591      double ptop = P(rz[i] + ctl->sample_dz);
06592      double pbot = P(rz[i] - ctl->sample_dz);
06593
06594      /* Init... */
06595      double mass = 0;
06596      int np = 0;
06597
06598      /* Loop over air parcels... */
06599 #pragma omp parallel for default(shared) reduction(+:mass,np)
06600      for (int ip = 0; ip < atm->np; ip++) {
06601
06602          /* Check time... */
06603          if (atm->time[ip] < t0 || atm->time[ip] > t1)
06604              continue;
06605
06606          /* Check latitude... */
06607          if (fabs(rlat[i] - atm->lat[ip]) > dlat)
06608              continue;
06609
06610          /* Check horizontal distance... */
06611          double x1[3];
06612          geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06613          if (DIST2(x0, x1) > rmax2)
06614              continue;
06615
06616          /* Check pressure... */
06617          if (ctl->sample_dz > 0)
06618              if (atm->p[ip] > pbot || atm->p[ip] < ptop)
06619                  continue;
06620
06621          /* Add mass... */
06622          if (ctl->qnt_m >= 0)
06623              mass += atm->q[ctl->qnt_m][ip];
06624          np++;
06625      }
06626
06627      /* Calculate column density... */
06628      double cd = mass / (1e6 * area);
06629
06630      /* Calculate volume mixing ratio... */
06631      double vmr = 0;
06632      if (ctl->molmass > 0 && ctl->sample_dz > 0) {
06633          if (mass > 0) {
06634
06635              /* Get temperature... */
06636              double temp;
06637              INTPOL_INIT;
06638              intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
06639                              rlon[i], rlat[i], &temp, ci, cw, 1);
06640
06641              /* Calculate volume mixing ratio... */
06642              vmr = MA / ctl->molmass * mass
06643                  / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
06644          }
06645      } else
06646          vmr = GSL_NAN;
06647
06648      /* Write output... */
06649      fprintf(out, "%.2f %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
06650              rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
06651  }
06652
06653      /* Finalize..... */
06654      if (t == ctl->t_stop) {
06655
06656          /* Close output file... */
06657          fclose(out);
06658
06659          /* Free... */
06660          free(rt);
06661          free(rz);
06662          free(rlon);
06663          free(rlat);
06664          free(robs);
06665      }
06666  }
06667
06668  /******
06669
06670 void write_station(
06671     const char *filename,
06672     ctl_t * ctl,
06673     atm_t * atm,
06674     double t) {
06675

```

```

06676 static FILE *out;
06677
06678 static double rmax2, x0[3], x1[3];
06679
06680 /* Set timer... */
06681 SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
06682
06683 /* Init... */
06684 if (t == ctl->t_start) {
06685
06686     /* Write info... */
06687     LOG(1, "Write station data: %s", filename);
06688
06689     /* Create new file... */
06690     if (!(out = fopen(filename, "w")))
06691         ERRMSG("Cannot create file!");
06692
06693     /* Write header... */
06694     fprintf(out,
06695             "# $1 = time [s]\n"
06696             "# $2 = altitude [km]\n"
06697             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
06698     for (int iq = 0; iq < ctl->nq; iq++)
06699         fprintf(out, "# $i = %s [%s]\n", (iq + 5),
06700             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
06701     fprintf(out, "\n");
06702
06703     /* Set geolocation and search radius... */
06704     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
06705     rmax2 = SQR(ctl->stat_r);
06706 }
06707
06708 /* Set time interval for output... */
06709 double t0 = t - 0.5 * ctl->dt_mod;
06710 double t1 = t + 0.5 * ctl->dt_mod;
06711
06712 /* Loop over air parcels... */
06713 for (int ip = 0; ip < atm->np; ip++) {
06714
06715     /* Check time... */
06716     if (atm->time[ip] < t0 || atm->time[ip] > t1)
06717         continue;
06718
06719     /* Check time range for station output... */
06720     if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
06721         continue;
06722
06723     /* Check station flag... */
06724     if (ctl->qnt_stat >= 0)
06725         if (atm->q[ctl->qnt_stat][ip])
06726             continue;
06727
06728     /* Get Cartesian coordinates... */
06729     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06730
06731     /* Check horizontal distance... */
06732     if (DIST2(x0, x1) > rmax2)
06733         continue;
06734
06735     /* Set station flag... */
06736     if (ctl->qnt_stat >= 0)
06737         atm->q[ctl->qnt_stat][ip] = 1;
06738
06739     /* Write data... */
06740     fprintf(out, "%.2f %g %g %g",
06741             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
06742     for (int iq = 0; iq < ctl->nq; iq++) {
06743         fprintf(out, " ");
06744         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
06745     }
06746     fprintf(out, "\n");
06747 }
06748
06749 /* Close file... */
06750 if (t == ctl->t_stop)
06751     fclose(out);
06752 }

```

## 5.21 libtrac.h File Reference

MPTRAC library declarations.

```
#include <ctype.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_statistics.h>
#include <math.h>
#include <netcdf.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
```

## Data Structures

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

## Macros

- #define [CPD](#) 1003.5  
*Specific heat of dry air at constant pressure [J/(kg K)].*
- #define [EPS](#) ([MH2O](#) / [MA](#))  
*Ratio of the specific gas constant of dry air and water vapor [1].*
- #define [G0](#) 9.80665  
*Standard gravity [m/s<sup>2</sup>].*
- #define [H0](#) 7.0  
*Scale height [km].*
- #define [LV](#) 2501000.  
*Latent heat of vaporization of water [J/kg].*
- #define [KB](#) 1.3806504e-23  
*Boltzmann constant [kg m<sup>2</sup>/(K s<sup>2</sup>)].*
- #define [MA](#) 28.9644  
*Molar mass of dry air [g/mol].*
- #define [MH2O](#) 18.01528  
*Molar mass of water vapor [g/mol].*
- #define [MO3](#) 48.00  
*Molar mass of ozone [g/mol].*
- #define [P0](#) 1013.25

- *Standard pressure [hPa].*
- #define **RA** (1e3 \* **RI** / **MA**)
- *Specific gas constant of dry air [J/(kg K)].*
- #define **RE** 6367.421
- *Mean radius of Earth [km].*
- #define **RI** 8.3144598
- *Ideal gas constant [J/(mol K)].*
- #define **T0** 273.15
- *Standard temperature [K].*
- #define **LEN** 5000
- *Maximum length of ASCII data lines.*
- #define **NP** 10000000
- *Maximum number of atmospheric data points.*
- #define **NQ** 15
- *Maximum number of quantities per data point.*
- #define **NCSI** 1000000
- *Maximum number of data points for CSI calculation.*
- #define **EP** 140
- *Maximum number of pressure levels for meteo data.*
- #define **EX** 1201
- *Maximum number of longitudes for meteo data.*
- #define **EY** 601
- *Maximum number of latitudes for meteo data.*
- #define **NENS** 2000
- *Maximum number of data points for ensemble analysis.*
- #define **NOBS** 10000000
- *Maximum number of observation data points.*
- #define **NTHREADS** 512
- *Maximum number of OpenMP threads.*
- #define **CY** 250
- *Maximum number of latitudes for climatological data.*
- #define **CP** 60
- *Maximum number of pressure levels for climatological data.*
- #define **CT** 12
- *Maximum number of time steps for climatological data.*
- #define **ALLOC**(ptr, type, n)
- *Allocate and clear memory.*
- #define **ARRAY\_2D**(ix, iy, ny) ((ix) \* (ny) + (iy))
- *Get 2-D array index.*
- #define **ARRAY\_3D**(ix, iy, ny, iz, nz) (((ix)\*(ny) + (iy)) \* (nz) + (iz))
- *Get 3-D array index.*
- #define **DEG2DX**(dlon, lat) ((dlon) \* **M\_PI** \* **RE** / 180. \* cos((lat) / 180. \* **M\_PI**))
- *Convert degrees to zonal distance.*
- #define **DEG2DY**(dlat) ((dlat) \* **M\_PI** \* **RE** / 180.)
- *Convert degrees to meridional distance.*
- #define **DP2DZ**(dp, p) (- (dp) \* **H0** / (p))
- *Convert pressure change to vertical distance.*
- #define **DX2DEG**(dx, lat)
- *Convert zonal distance to degrees.*
- #define **DY2DEG**(dy) ((dy) \* 180. / (**M\_PI** \* **RE**))
- *Convert meridional distance to degrees.*

- #define **DZ2DP**(dz, p)  $-(dz) * (p) / H0$   
*Convert vertical distance to pressure change.*
- #define **DIST**(a, b)  $\text{sqrt}(\text{DIST2}(a, b))$   
*Compute Cartesian distance between two vectors.*
- #define **DIST2**(a, b)  $((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))$   
*Compute squared distance between two vectors.*
- #define **DOTP**(a, b)  $a[0]*b[0]+a[1]*b[1]+a[2]*b[2]$   
*Compute dot product of two vectors.*
- #define **FMOD**(x, y)  $((x) - (\text{int})((x) / (y)) * (y))$   
*Compute floating point modulo.*
- #define **FREAD**(ptr, type, size, out)  
*Read binary data.*
- #define **FWRITE**(ptr, type, size, out)  
*Write binary data.*
- #define **INTPOL\_INIT** double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};  
*Initialize cache variables for interpolation.*
- #define **INTPOL\_2D**(var, init)  
*2-D interpolation of a meteo variable.*
- #define **INTPOL\_3D**(var, init)  
*3-D interpolation of a meteo variable.*
- #define **INTPOL\_SPACE\_ALL**(p, lon, lat)  
*Spatial interpolation of all meteo data.*
- #define **INTPOL\_TIME\_ALL**(time, p, lon, lat)  
*Temporal interpolation of all meteo data.*
- #define **LAPSE**(p1, t1, p2, t2)  
*Calculate lapse rate between pressure levels.*
- #define **LIN**(x0, y0, x1, y1, x)  $((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))$   
*Compute linear interpolation.*
- #define **NC**(cmd)  
*Execute netCDF library command and check result.*
- #define **NC\_DEF\_VAR**(varname, type, ndims, dims, long\_name, units)  
*Define netCDF variable.*
- #define **NC\_GET\_DOUBLE**(varname, ptr, force)  
*Read netCDF double array.*
- #define **NC\_INQ\_DIM**(dimname, ptr, min, max)  
*Read netCDF dimension.*
- #define **NC\_PUT\_DOUBLE**(varname, ptr, hyperslab)  
*Write netCDF double array.*
- #define **NC\_PUT\_INT**(varname, ptr, hyperslab)  
*Write netCDF integer array.*
- #define **NC\_PUT\_ATT**(attname, text) **NC**(nc\_put\_att\_text(ncid, NC\_GLOBAL, attname, strlen(text), text));  
*Set netCDF global attribute.*
- #define **NC\_PUT\_FLOAT**(varname, ptr, hyperslab)  
*Write netCDF float array.*
- #define **NN**(x0, y0, x1, y1, x)  $(\text{fabs}((x) - (x0)) \leq \text{fabs}((x) - (x1)) ? (y0) : (y1))$   
*Compute nearest neighbor interpolation.*
- #define **NORM**(a)  $\text{sqrt}(\text{DOTP}(a, a))$   
*Compute norm of a vector.*
- #define **P**(z)  $(P0 * \exp(-(z) / H0))$   
*Convert altitude to pressure.*
- #define **PSAT**(t)  $(6.112 * \exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))$

- *Compute saturation pressure over water (WMO, 2018).*  
 • #define **PSICE**(t) (0.01 \* pow(10., -2663.5 / (t) + 12.537))
- *Compute saturation pressure over ice (Marti and Mauersberger, 1993).*  
 • #define **PW**(p, h2o)  
   *Calculate partial water vapor pressure.*
- #define **RH**(p, t, h2o) (**PW**(p, h2o) / **PSAT**(t) \* 100.)  
   *Compute relative humidity over water.*
- #define **RHICE**(p, t, h2o) (**PW**(p, h2o) / **PSICE**(t) \* 100.)  
   *Compute relative humidity over ice.*
- #define **RHO**(p, t) (100. \* (p) / (**RA** \* (t)))  
   *Compute density of air.*
- #define **SET\_ATM**(qnt, val)  
   *Set atmospheric quantity value.*
- #define **SET\_QNT**(qnt, name, longname, unit)  
   *Set atmospheric quantity index.*
- #define **SH**(h2o) (**EPS** \* **GSL\_MAX**((h2o), 0.1e-6))  
   *Compute specific humidity from water vapor volume mixing ratio.*
- #define **SQR**(x) ((x)\*(x))  
   *Compute square.*
- #define **SWAP**(x, y, type) do {type tmp = x; x = y; y = tmp;} while(0);  
   *Swap macro.*
- #define **TDEW**(p, h2o)  
   *Calculate dew point temperature (WMO, 2018).*
- #define **TICE**(p, h2o) (-2663.5 / (log10(100. \* **PW**((p), (h2o)))) - 12.537))  
   *Calculate frost point temperature (Marti and Mauersberger, 1993).*
- #define **THETA**(p, t) ((t) \* pow(1000. / (p), 0.286))  
   *Compute potential temperature.*
- #define **THETA\_VIRT**(p, t, h2o) (**TVIRT**(**THETA**((p), (t)), **GSL\_MAX**((h2o), 0.1e-6)))  
   *Compute virtual potential temperature.*
- #define **TOK**(line, tok, format, var)  
   *Get string tokens.*
- #define **TVIRT**(t, h2o) ((t) \* (1. + (1. - **EPS**) \* **GSL\_MAX**((h2o), 0.1e-6)))  
   *Compute virtual temperature.*
- #define **Z**(p) (**H0** \* log(**P0** / (p)))  
   *Convert pressure to altitude.*
- #define **ZDIFF**(lnp0, t0, h2o0, lnp1, t1, h2o1)  
   *Calculate geopotential height difference.*
- #define **ZETA**(ps, p, t)  
   *Calculate zeta vertical coordinate.*
- #define **LOGLEV** 2  
   *Level of log messages (0=none, 1=basic, 2=detailed, 3=debug).*
- #define **LOG**(level, ...)  
   *Print log message.*
- #define **WARN**(...)  
   *Print warning message.*
- #define **ERRMSG**(...)  
   *Print error message and quit program.*
- #define **PRINT**(format, var)  
   *Print macro for debugging.*
- #define **NTIMER** 100  
   *Maximum number of timers.*

- `#define PRINT_TIMERS timer("END", "END", 1);`  
*Print timers.*
- `#define SELECT_TIMER(id, group, color)`  
*Select timer.*
- `#define START_TIMERS NVTX_PUSH("START", NVTX_CPU);`  
*Start timers.*
- `#define STOP_TIMERS NVTX_POP;`  
*Stop timers.*
- `#define NVTX_PUSH(range_title, range_color) {}`
- `#define NVTX_POP {}`

## Functions

- `void thrustSortWrapper (double * __restrict__ c, int n, int * __restrict__ index)`  
*Wrapper to Thrust sorting function.*
- `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 (clim_t *clim, double t, double lat, double p)`  
*Climatology of HNO3 volume mixing ratios.*
- `void clim_hno3_init (clim_t *clim)`  
*Initialization function for HNO3 climatology.*
- `double clim_oh (clim_t *clim, double t, double lat, double p)`  
*Climatology of OH number concentrations.*
- `double clim_oh_diurnal (ctl_t *ctl, clim_t *clim, double t, double p, double lon, double lat)`  
*Climatology of OH number concentrations with diurnal variation.*
- `void clim_oh_init (ctl_t *ctl, clim_t *clim)`  
*Initialization function for OH climatology.*
- `double clim_oh_init_help (double beta, double time, double lat)`  
*Apply diurnal correction to OH climatology.*
- `double clim_h2o2 (clim_t *clim, double t, double lat, double p)`  
*Climatology of H2O2 number concentrations.*
- `void clim_h2o2_init (ctl_t *ctl, clim_t *clim)`  
*Initialization function for H2O2 climatology.*
- `double clim_tropo (clim_t *clim, double t, double lat)`  
*Climatology of tropopause pressure.*
- `void clim_tropo_init (clim_t *clim)`  
*Initialize tropopause climatology.*
- `void compress_pack (char *varname, float *array, size_t nxy, size_t nz, int decompress, FILE *inout)`  
*Pack or unpack array.*
- `void day2doy (int year, int mon, int day, int *doy)`  
*Compress or decompress array with zfp.*
- `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, clim_t *clim, double t, met_t **met0, met_t **met1)`  
*Get meteo data for given time step.*
- `void get_met_help (ctl_t *ctl, double t, int direct, char *metbase, double dt_met, char *filename)`



- Get meteo 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 meteo 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 meteo 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)

*Spatial interpolation of meteo 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 meteo data.*
- void `jsec2time` (double jsec, int \*year, int \*mon, int \*day, int \*hour, int \*min, int \*sec, double \*remain)

*Temporal interpolation of meteo data.*
- 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.*
- void `quicksort` (double arr[], int brr[], int low, int high)

*Parallel quicksort.*
- int `quicksort_partition` (double arr[], int brr[], int low, int high)

*Partition function for quicksort.*
- int `read_atm` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)

*Read atmospheric data.*
- int `read_atm_asc` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)

*Read atmospheric data in ASCII format.*
- int `read_atm_bin` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)

*Read atmospheric data in binary format.*
- int `read_atm_clams` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)

*Read atmospheric data in CLaMS format.*
- int `read_atm_nc` (const char \*filename, ctl\_t \*ctl, atm\_t \*atm)

*Read atmospheric data in netCDF format.*
- void `read_clim` (ctl\_t \*ctl, clim\_t \*clim)

*Read climatological data.*
- void `read_ctl` (const char \*filename, int argc, char \*argv[], ctl\_t \*ctl)

*Read control parameters.*
- int `read_met` (char \*filename, ctl\_t \*ctl, clim\_t \*clim, met\_t \*met)

*Read meteo data file.*
- void `read_met_bin_2d` (FILE \*out, met\_t \*met, float var[EX][EY], char \*varname)

*Read 2-D meteo variable.*
- void `read_met_bin_3d` (FILE \*in, ctl\_t \*ctl, met\_t \*met, float var[EX][EY][EP], char \*varname, int precision, double tolerance)

*Read 3-D meteo variable.*
- void `read_metCAPE` (clim\_t \*clim, met\_t \*met)

- Calculate convective available potential energy.*
- void `read_met_cloud` (`ctl_t *ctl`, `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 meteo 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 meteo data.*
- void `read_met_levels` (`int ncid`, `ctl_t *ctl`, `met_t *met`)
- Read meteo data on vertical levels.*
- void `read_met_ml2pl` (`ctl_t *ctl`, `met_t *met`, `float var[EX][EY][EP]`)
- Convert meteo data from model levels to pressure levels.*
- int `read_met_nc_2d` (`int ncid`, `char *varname`, `char *varname2`, `ctl_t *ctl`, `met_t *met`, `float dest[EX][EY]`, `float scl`, `int init`)
- Read and convert 2D variable from meteo data file.*
- int `read_met_nc_3d` (`int ncid`, `char *varname`, `char *varname2`, `ctl_t *ctl`, `met_t *met`, `float dest[EX][EY][EP]`, `float scl`, `int init`)
- Read and convert 3D variable from meteo data file.*
- void `read_met_pbl` (`met_t *met`)
- Calculate pressure of the boundary layer.*
- void `read_met_periodic` (`met_t *met`)
- Create meteo 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 meteo data.*
- void `read_met_surface` (`int ncid`, `met_t *met`, `ctl_t *ctl`)
- Read surface data.*
- void `read_met_tropo` (`ctl_t *ctl`, `clim_t *clim`, `met_t *met`)
- Calculate tropopause data.*
- void `read_obs` (`char *filename`, `double *rt`, `double *rz`, `double *rlon`, `double *rlat`, `double *robs`, `int *nobs`)
- Read observation 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 rp`, `double rhop`)
- Calculate sedimentation velocity.*
- void `spline` (`double *x`, `double *y`, `int n`, `double *x2`, `double *y2`, `int n2`, `int method`)
- Spline interpolation.*
- float `stddev` (`float *data`, `int n`)
- Calculate standard deviation.*
- double `sza` (`double sec`, `double lon`, `double lat`)
- Calculate solar zenith angle.*
- 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`, `const char *group`, `int output`)
- Measure wall-clock time.*
- double `tropo_weight` (`clim_t *clim`, `double t`, `double lat`, `double p`)

- Get weighting factor based on tropopause distance.*
- void [write\\_atm](#) (const char \*filename, [ctl\\_t](#) \*ctl, [atm\\_t](#) \*atm, double t)  
*Write atmospheric data.*
- void [write\\_atm\\_asc](#) (const char \*filename, [ctl\\_t](#) \*ctl, [atm\\_t](#) \*atm, double t)  
*Write atmospheric data in ASCII format.*
- void [write\\_atm\\_bin](#) (const char \*filename, [ctl\\_t](#) \*ctl, [atm\\_t](#) \*atm)  
*Write atmospheric data in binary format.*
- void [write\\_atm\\_clams](#) ([ctl\\_t](#) \*ctl, [atm\\_t](#) \*atm, double t)  
*Write atmospheric data in CLaMS format.*
- void [write\\_atm\\_nc](#) (const char \*filename, [ctl\\_t](#) \*ctl, [atm\\_t](#) \*atm)  
*Write atmospheric data in netCDF format.*
- 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\\_grid\\_asc](#) (const char \*filename, [ctl\\_t](#) \*ctl, double \*cd, double \*vmr\_expl, double \*vmr\_impl, double t, double \*z, double \*lon, double \*lat, double \*area, double dz, int \*np)  
*Write gridded data in ASCII format.*
- void [write\\_grid\\_nc](#) (const char \*filename, [ctl\\_t](#) \*ctl, double \*cd, double \*vmr\_expl, double \*vmr\_impl, double t, double \*z, double \*lon, double \*lat, double \*area, double dz, int \*np)  
*Write gridded data in netCDF format.*
- int [write\\_met](#) (char \*filename, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met)  
*Read meteo data file.*
- void [write\\_met\\_bin\\_2d](#) (FILE \*out, [met\\_t](#) \*met, float var[EX][EY], char \*varname)  
*Write 2-D meteo variable.*
- void [write\\_met\\_bin\\_3d](#) (FILE \*out, [ctl\\_t](#) \*ctl, [met\\_t](#) \*met, float var[EX][EY][EP], char \*varname, int precision, double tolerance)  
*Write 3-D meteo variable.*
- 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 declarations.

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

### 5.21.2 Macro Definition Documentation

**5.21.2.1 CPD** `#define CPD 1003.5`

Specific heat of dry air at constant pressure [J/(kg K)].

Definition at line 83 of file [libtrac.h](#).

**5.21.2.2 EPS** `#define EPS (MH2O / MA)`

Ratio of the specific gas constant of dry air and water vapor [1].

Definition at line 88 of file [libtrac.h](#).

**5.21.2.3 G0** `#define G0 9.80665`

Standard gravity [m/s<sup>2</sup>].

Definition at line 93 of file [libtrac.h](#).

**5.21.2.4 H0** `#define H0 7.0`

Scale height [km].

Definition at line 98 of file [libtrac.h](#).

**5.21.2.5 LV** `#define LV 2501000.`

Latent heat of vaporization of water [J/kg].

Definition at line 103 of file [libtrac.h](#).

**5.21.2.6 KB** `#define KB 1.3806504e-23`

Boltzmann constant [kg m<sup>2</sup>/(K s<sup>2</sup>)].

Definition at line 108 of file [libtrac.h](#).

**5.21.2.7 MA** `#define MA 28.9644`

Molar mass of dry air [g/mol].

Definition at line 113 of file [libtrac.h](#).

**5.21.2.8 MH2O** `#define MH2O 18.01528`

Molar mass of water vapor [g/mol].

Definition at line 118 of file [libtrac.h](#).

**5.21.2.9 MO3** `#define MO3 48.00`

Molar mass of ozone [g/mol].

Definition at line 123 of file [libtrac.h](#).

**5.21.2.10 P0** `#define P0 1013.25`

Standard pressure [hPa].

Definition at line 128 of file [libtrac.h](#).

**5.21.2.11 RA** `#define RA (1e3 * RI / MA)`

Specific gas constant of dry air [J/(kg K)].

Definition at line 133 of file [libtrac.h](#).

**5.21.2.12 RE** `#define RE 6367.421`

Mean radius of Earth [km].

Definition at line 138 of file [libtrac.h](#).

**5.21.2.13 RI** `#define RI 8.3144598`

Ideal gas constant [J/(mol K)].

Definition at line [143](#) of file [libtrac.h](#).

**5.21.2.14 T0** `#define T0 273.15`

Standard temperature [K].

Definition at line [148](#) of file [libtrac.h](#).

**5.21.2.15 LEN** `#define LEN 5000`

Maximum length of ASCII data lines.

Definition at line [157](#) of file [libtrac.h](#).

**5.21.2.16 NP** `#define NP 10000000`

Maximum number of atmospheric data points.

Definition at line [162](#) of file [libtrac.h](#).

**5.21.2.17 NQ** `#define NQ 15`

Maximum number of quantities per data point.

Definition at line [167](#) of file [libtrac.h](#).

**5.21.2.18 NCSI** `#define NCSI 1000000`

Maximum number of data points for CSI calculation.

Definition at line [172](#) of file [libtrac.h](#).

**5.21.2.19 EP** `#define EP 140`

Maximum number of pressure levels for meteo data.

Definition at line 177 of file [libtrac.h](#).

**5.21.2.20 EX** `#define EX 1201`

Maximum number of longitudes for meteo data.

Definition at line 182 of file [libtrac.h](#).

**5.21.2.21 EY** `#define EY 601`

Maximum number of latitudes for meteo data.

Definition at line 187 of file [libtrac.h](#).

**5.21.2.22 NENS** `#define NENS 2000`

Maximum number of data points for ensemble analysis.

Definition at line 192 of file [libtrac.h](#).

**5.21.2.23 NOBS** `#define NOBS 10000000`

Maximum number of observation data points.

Definition at line 197 of file [libtrac.h](#).

**5.21.2.24 NTHREADS** `#define NTHREADS 512`

Maximum number of OpenMP threads.

Definition at line 202 of file [libtrac.h](#).

**5.21.2.25 CY** `#define CY 250`

Maximum number of latitudes for climatological data.

Definition at line 207 of file [libtrac.h](#).

**5.21.2.26 CP** `#define CP 60`

Maximum number of pressure levels for climatological data.

Definition at line 212 of file [libtrac.h](#).

**5.21.2.27 CT** `#define CT 12`

Maximum number of time steps for climatological data.

Definition at line 217 of file [libtrac.h](#).

**5.21.2.28 ALLOC** `#define ALLOC(  
 ptr,  
 type,  
 n )`**Value:**

```
if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
    ERRMSG("Out of memory!");
```

Allocate and clear memory.

Definition at line 232 of file [libtrac.h](#).

**5.21.2.29 ARRAY\_2D** `#define ARRAY_2D(  
 ix,  
 iy,  
 ny ) ((ix) * (ny) + (iy))`

Get 2-D array index.

Definition at line 238 of file [libtrac.h](#).



**5.21.2.30 ARRAY\_3D** `#define ARRAY_3D(  
     ix,  
     iy,  
     ny,  
     iz,  
     nz )   (((ix)*(ny) + (iy)) * (nz) + (iz))`

Get 3-D array index.

Definition at line 242 of file [libtrac.h](#).

**5.21.2.31 DEG2DX** `#define DEG2DX(  
     dlon,  
     lat )   ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))`

Convert degrees to zonal distance.

Definition at line 246 of file [libtrac.h](#).

**5.21.2.32 DEG2DY** `#define DEG2DY(  
     dlat )   ((dlat) * M_PI * RE / 180.)`

Convert degrees to meridional distance.

Definition at line 250 of file [libtrac.h](#).

**5.21.2.33 DP2DZ** `#define DP2DZ(  
     dp,  
     p )   (- (dp) * H0 / (p))`

Convert pressure change to vertical distance.

Definition at line 254 of file [libtrac.h](#).

**5.21.2.34 DX2DEG** `#define DX2DEG(  
     dx,  
     lat )`

**Value:**

```
((lat) < -89.999 || (lat) > 89.999) ? 0 \
: (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI))
```

Convert zonal distance to degrees.

Definition at line 258 of file [libtrac.h](#).

**5.21.2.35 DY2DEG** `#define DY2DEG(  
 dy ) ((dy) * 180. / (M_PI * RE))`

Convert meridional distance to degrees.

Definition at line 263 of file [libtrac.h](#).

**5.21.2.36 DZ2DP** `#define DZ2DP(  
 dz,  
 p ) (-(dz) * (p) / H0)`

Convert vertical distance to pressure change.

Definition at line 267 of file [libtrac.h](#).

**5.21.2.37 DIST** `#define DIST(  
 a,  
 b ) sqrt(DIST2(a, b))`

Compute Cartesian distance between two vectors.

Definition at line 271 of file [libtrac.h](#).

**5.21.2.38 DIST2** `#define DIST2(  
 a,  
 b ) ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))`

Compute squared distance between two vectors.

Definition at line 275 of file [libtrac.h](#).

**5.21.2.39 DOTP** `#define DOTP(  
 a,  
 b ) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])`

Compute dot product of two vectors.

Definition at line 279 of file [libtrac.h](#).

**5.21.2.40 FMOD** `#define FMOD(  
     x,  
     y )   ((x) - (int) ((x) / (y)) * (y))`

Compute floating point modulo.

Definition at line 283 of file [libtrac.h](#).

**5.21.2.41 FREAD** `#define FREAD(  
     ptr,  
     type,  
     size,  
     out )`

**Value:**

```
{
    if(fread(ptr, sizeof(type), size, out)!=size)
        ERRMSG("Error while reading!");
}
```

Read binary data.

Definition at line 287 of file [libtrac.h](#).

**5.21.2.42 FWRITE** `#define FWRITE(  
     ptr,  
     type,  
     size,  
     out )`

**Value:**

```
{
    if(fwrite(ptr, sizeof(type), size, out)!=size)
        ERRMSG("Error while writing!");
}
```

Write binary data.

Definition at line 293 of file [libtrac.h](#).

**5.21.2.43 INTPOL\_INIT** `#define INTPOL_INIT   double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};`

Initialize cache variables for interpolation.

Definition at line 299 of file [libtrac.h](#).

**5.21.2.44 INTPOL\_2D** #define INTPOL\_2D(

```
var,
init )
```

**Value:**

```
intpol_met_time_2d(met0, met0->var, met1, met1->var,
                  atm->time[ip], atm->lon[ip], atm->lat[ip],
                  &var, ci, cw, init);
```

2-D interpolation of a meteo variable.

Definition at line 303 of file [libtrac.h](#).

**5.21.2.45 INTPOL\_3D** #define INTPOL\_3D(

```
var,
init )
```

**Value:**

```
intpol_met_time_3d(met0, met0->var, met1, met1->var,
                  atm->time[ip], atm->p[ip],
                  atm->lon[ip], atm->lat[ip],
                  &var, ci, cw, init);
```

3-D interpolation of a meteo variable.

Definition at line 309 of file [libtrac.h](#).

**5.21.2.46 INTPOL\_SPACE\_ALL** #define INTPOL\_SPACE\_ALL(

```
p,
lon,
lat )
```

**Value:**

```
{
  intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1);
  intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0);
  intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0);
  intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0);
  intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0);
  intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0);
  intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0);
  intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0);
  intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0);
  intpol_met_space_3d(met, met->iwc, p, lon, lat, &iwc, ci, cw, 0);
  intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
  intpol_met_space_2d(met, met->ts, lon, lat, &ts, ci, cw, 0);
  intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0);
  intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0);
  intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0);
  intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0);
  intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
  intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0);
  intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0);
  intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0);
  intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0);
  intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0);
  intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0);
  intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0);
  intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0);
  intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0);
  intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0);
  intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0);
}
```

Spatial interpolation of all meteo data.

Definition at line 316 of file [libtrac.h](#).

**5.21.2.47 INTPOL\_TIME\_ALL** #define INTPOL\_TIME\_ALL(

```

    time,
    p,
    lon,
    lat )

```

**Value:**

```

{
    \
    intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
    intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->cl, met1, met1->cl, time, lon, lat, &cl, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0); \
}

```

Temporal interpolation of all meteo data.

Definition at line 348 of file [libtrac.h](#).**5.21.2.48 LAPSE** #define LAPSE(

```

    p1,
    t1,
    p2,
    t2 )

```

**Value:**

```

(1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1))
 * ((p2) + (p1)) / ((p2) - (p1))) \

```

Calculate lapse rate between pressure levels.

Definition at line 380 of file [libtrac.h](#).**5.21.2.49 LIN** #define LIN(

```

    x0,
    y0,
    x1,
    y1,
    x ) ((y0) + ((y1) - (y0)) / ((x1) - (x0)) * ((x) - (x0)))

```

Compute linear interpolation.

Definition at line 385 of file [libtrac.h](#).

**5.21.2.50 NC** `#define NC(`  
`cmd )`

**Value:**

```
{
    int nc_result=(cmd);
    if(nc_result!=NC_NOERR)
        ERRMSG("%s", nc_strerror(nc_result));
}
```

Execute netCDF library command and check result.

Definition at line 389 of file [libtrac.h](#).

**5.21.2.51 NC\_DEF\_VAR** `#define NC_DEF_VAR(`  
`varname,`  
`type,`  
`ndims,`  
`dims,`  
`long_name,`  
`units )`

**Value:**

```
{
    NC(nc_def_var(ncid, varname, type, ndims, dims, &varid));
    NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));
    NC(nc_put_att_text(ncid, varid, "units", strlen(units), units));
}
```

Define netCDF variable.

Definition at line 396 of file [libtrac.h](#).

**5.21.2.52 NC\_GET\_DOUBLE** `#define NC_GET_DOUBLE(`  
`varname,`  
`ptr,`  
`force )`

**Value:**

```
{
    if(force) {
        NC(nc_inq_varid(ncid, varname, &varid));
        NC(nc_get_var_double(ncid, varid, ptr));
    } else {
        if(nc_inq_varid(ncid, varname, &varid) == NC_NOERR) {
            NC(nc_get_var_double(ncid, varid, ptr));
        } else
            WARN("netCDF variable %s is missing!", varname);
    }
}
```

Read netCDF double array.

Definition at line 403 of file [libtrac.h](#).

**5.21.2.53 NC\_INQ\_DIM** `#define NC_INQ_DIM(`  
     *dimname,*  
     *ptr,*  
     *min,*  
     *max )*

**Value:**

```
{
    \
    int dimid; size_t naux;
    NC(nc_inq_dimid(ncid, dimname, &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &naux));
    *ptr = (int)naux;
    if ((*ptr) < (min) || (*ptr) > (max))
        ERRMSG("Dimension %s is out of range!", dimname); \
}
```

Read netCDF dimension.

Definition at line 416 of file [libtrac.h](#).

**5.21.2.54 NC\_PUT\_DOUBLE** `#define NC_PUT_DOUBLE(`  
     *varname,*  
     *ptr,*  
     *hyperslab )*

**Value:**

```
{
    \
    NC(nc_inq_varid(ncid, varname, &varid));
    if(hyperslab) {
        NC(nc_put_vara_double(ncid, varid, start, count, ptr));
    } else {
        NC(nc_put_var_double(ncid, varid, ptr));
    }
}
```

Write netCDF double array.

Definition at line 426 of file [libtrac.h](#).

**5.21.2.55 NC\_PUT\_INT** `#define NC_PUT_INT(`  
     *varname,*  
     *ptr,*  
     *hyperslab )*

**Value:**

```
{
    \
    NC(nc_inq_varid(ncid, varname, &varid));
    if(hyperslab) {
        NC(nc_put_vara_int(ncid, varid, start, count, ptr));
    } else {
        NC(nc_put_var_int(ncid, varid, ptr));
    }
}
```

Write netCDF integer array.

Definition at line 436 of file [libtrac.h](#).

```

5.21.2.56 NC_PUT_ATT #define NC_PUT_ATT(
    attname,
    text ) NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));

```

Set netCDF global attribute.

Definition at line 446 of file [libtrac.h](#).

```

5.21.2.57 NC_PUT_FLOAT #define NC_PUT_FLOAT(
    varname,
    ptr,
    hyperslab )

```

Value:

```

{
    NC(nc_inq_varid(ncid, varname, &varid));
    if(hyperslab) {
        NC(nc_put_vara_float(ncid, varid, start, count, ptr));
    } else {
        NC(nc_put_var_float(ncid, varid, ptr));
    }
}

```

Write netCDF float array.

Definition at line 450 of file [libtrac.h](#).

```

5.21.2.58 NN #define NN(
    x0,
    y0,
    x1,
    y1,
    x ) (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))

```

Compute nearest neighbor interpolation.

Definition at line 460 of file [libtrac.h](#).

```

5.21.2.59 NORM #define NORM(
    a ) sqrt(DOTP(a, a))

```

Compute norm of a vector.

Definition at line 464 of file [libtrac.h](#).



**5.21.2.60 P** `#define P(  
z ) (P0 * exp(-(z) / H0))`

Convert altitude to pressure.

Definition at line 468 of file [libtrac.h](#).

**5.21.2.61 PSAT** `#define PSAT(  
t ) (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))`

Compute saturation pressure over water (WMO, 2018).

Definition at line 472 of file [libtrac.h](#).

**5.21.2.62 PSICE** `#define PSICE(  
t ) (0.01 * pow(10., -2663.5 / (t) + 12.537))`

Compute saturation pressure over ice (Marti and Mauersberger, 1993).

Definition at line 476 of file [libtrac.h](#).

**5.21.2.63 PW** `#define PW(  
p,  
h2o )`

**Value:**

```
((p) * GSL_MAX((h2o), 0.1e-6) \
/ (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
```

Calculate partial water vapor pressure.

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

**5.21.2.64 RH** `#define RH(  
p,  
t,  
h2o ) (PW(p, h2o) / PSAT(t) * 100.)`

Compute relative humidity over water.

Definition at line 485 of file [libtrac.h](#).

**5.21.2.65 RHICE** `#define RHICE(  
 p,  
 t,  
 h2o ) (PW(p, h2o) / PSICE(t) * 100.)`

Compute relative humidity over ice.

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

**5.21.2.66 RHO** `#define RHO(  
 p,  
 t ) (100. * (p) / (RA * (t)))`

Compute density of air.

Definition at line 493 of file [libtrac.h](#).

**5.21.2.67 SET\_ATM** `#define SET_ATM(  
 qnt,  
 val )`

**Value:**

```
if (ctl->qnt >= 0) \
    atm->q[ctl->qnt][ip] = val;
```

Set atmospheric quantity value.

Definition at line 497 of file [libtrac.h](#).

**5.21.2.68 SET\_QNT** `#define SET_QNT(  
 qnt,  
 name,  
 longname,  
 unit )`

**Value:**

```
if (strcasecmp(ctl->qnt_name[iq], name) == 0) { \
    ctl->qnt = iq; \
    sprintf(ctl->qnt_longname[iq], longname); \
    sprintf(ctl->qnt_unit[iq], unit); \
} else
```

Set atmospheric quantity index.

Definition at line 502 of file [libtrac.h](#).

**5.21.2.69 SH** `#define SH(  
h2o ) (EPS * GSL_MAX((h2o), 0.1e-6))`

Compute specific humidity from water vapor volume mixing ratio.

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

**5.21.2.70 SQR** `#define SQR(  
x ) ((x)*(x))`

Compute square.

Definition at line 514 of file [libtrac.h](#).

**5.21.2.71 SWAP** `#define SWAP(  
x,  
y,  
type ) do {type tmp = x; x = y; y = tmp;} while(0);`

Swap macro.

Definition at line 518 of file [libtrac.h](#).

**5.21.2.72 TDEW** `#define TDEW(  
p,  
h2o )`

**Value:**

$$\frac{(T0 + 243.12 * \log(PW((p), (h2o)) / 6.112))}{(17.62 - \log(PW((p), (h2o)) / 6.112))} \quad \backslash$$

Calculate dew point temperature (WMO, 2018).

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

**5.21.2.73 TICE** `#define TICE(  
p,  
h2o ) (-2663.5 / (log10(100. * PW((p), (h2o))) - 12.537))`

Calculate frost point temperature (Marti and Mauersberger, 1993).

Definition at line 527 of file [libtrac.h](#).

```
5.21.2.74 THETA #define THETA(
    p,
    t ) ((t) * pow(1000. / (p), 0.286))
```

Compute potential temperature.

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

```
5.21.2.75 THETA_VIRT #define THETA_VIRT(
    p,
    t,
    h2o ) (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))
```

Compute virtual potential temperature.

Definition at line 535 of file [libtrac.h](#).

```
5.21.2.76 TOK #define TOK(
    line,
    tok,
    format,
    var )
```

Value:

```
{
    if(((tok)=strtok((line), " \t"))) {
        if(sscanf(tok, format, &(var))!=1) continue;
    } else ERRMSG("Error while reading!");
}
```

Get string tokens.

Definition at line 539 of file [libtrac.h](#).

```
5.21.2.77 TVIRT #define TVIRT(
    t,
    h2o ) ((t) * (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
```

Compute virtual temperature.

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

```
5.21.2.78 Z #define Z(
    p ) (H0 * log(P0 / (p)))
```

Convert pressure to altitude.

Definition at line 550 of file [libtrac.h](#).

**5.21.2.79 ZDIFF** `#define ZDIFF(`

```

    lnp0,
    t0,
    h2o0,
    lnp1,
    t1,
    h2o1 )

```

**Value:**

```

(RI / MA / GO * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1))) \
 * ((lnp0) - (lnp1)))

```

Calculate geopotential height difference.

Definition at line 554 of file [libtrac.h](#).

**5.21.2.80 ZETA** `#define ZETA(`

```

    ps,
    p,
    t )

```

**Value:**

```

((p) / (ps) <= 0.3 ? 1. : \
 sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3))) \
 * THETA((p), (t)))

```

Calculate zeta vertical coordinate.

Definition at line 559 of file [libtrac.h](#).

**5.21.2.81 LOGLEV** `#define LOGLEV 2`

Level of log messages (0=none, 1=basic, 2=detailed, 3=debug).

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

**5.21.2.82 LOG** `#define LOG(`

```

    level,
    ... )

```

**Value:**

```

{ \
 if (level >= 2) \
     printf(" "); \
 if (level <= LOGLEV) { \
     printf(__VA_ARGS__); \
     printf("\n"); \
 } \
}

```

Print log message.

Definition at line 574 of file [libtrac.h](#).

**5.21.2.83 WARN** `#define WARN(  
 ... )`

**Value:**

```
{  
    printf("\nWarning (%s, %s, l%d): ", __FILE__, __func__, __LINE__); \  
    LOG(0, __VA_ARGS__); \  
}
```

Print warning message.

Definition at line [584](#) of file [libtrac.h](#).

**5.21.2.84 ERRMSG** `#define ERRMSG(  
 ... )`

**Value:**

```
{  
    printf("\nError (%s, %s, l%d): ", __FILE__, __func__, __LINE__); \  
    LOG(0, __VA_ARGS__); \  
    exit(EXIT_FAILURE); \  
}
```

Print error message and quit program.

Definition at line [590](#) of file [libtrac.h](#).

**5.21.2.85 PRINT** `#define PRINT(  
 format,  
 var )`

**Value:**

```
printf("Print (%s, %s, l%d): %s= \"format\"\n", \  
    __FILE__, __func__, __LINE__, #var, var);
```

Print macro for debugging.

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

**5.21.2.86 NTIMER** `#define NTIMER 100`

Maximum number of timers.

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

**5.21.2.87 PRINT\_TIMERS** `#define PRINT_TIMERS timer("END", "END", 1);`

Print timers.

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

**5.21.2.88 SELECT\_TIMER** `#define SELECT_TIMER(  
 id,  
 group,  
 color )`

**Value:**

```
{  
    NVTX_POP; \  
    NVTX_PUSH(id, color); \  
    timer(id, group, 0); \  
}
```

Select timer.

Definition at line 613 of file [libtrac.h](#).

**5.21.2.89 START\_TIMERS** `#define START_TIMERS NVTX_PUSH("START", NVTX_CPU);`

Start timers.

Definition at line 620 of file [libtrac.h](#).

**5.21.2.90 STOP\_TIMERS** `#define STOP_TIMERS NVTX_POP;`

Stop timers.

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

**5.21.2.91 NVTX\_PUSH** `#define NVTX_PUSH(  
 range_title,  
 range_color ) {}`

Definition at line 671 of file [libtrac.h](#).

**5.21.2.92 NVTX\_POP** `#define NVTX_POP {}`

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

## 5.21.3 Function Documentation

**5.21.3.1 thrustSortWrapper()** void thrustSortWrapper (

```
double *__restrict__ c,
int n,
int *__restrict__ index )
```

Wrapper to Thrust sorting function.

**5.21.3.2 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 }
```

**5.21.3.3 check\_finite()** int check\_finite (

```
const double x )
```

Check if x is finite.

**5.21.3.4 clim\_hno3()** double clim\_hno3 (

```
clim_t * clim,
double t,
double lat,
double p )
```

Climatology of HNO3 volume mixing ratios.

Definition at line 43 of file [libtrac.c](#).

```
00047     {
00048
00049     /* Get seconds since begin of year... */
00050     double sec = FMOD(t, 365.25 * 86400.);
00051     while (sec < 0)
00052         sec += 365.25 * 86400.;
00053
00054     /* Check pressure... */
00055     if (p < clim->hno3_p[0])
00056         p = clim->hno3_p[0];
00057     else if (p > clim->hno3_p[clim->hno3_np - 1])
00058         p = clim->hno3_p[clim->hno3_np - 1];
00059
00060     /* Check latitude... */
00061     if (lat < clim->hno3_lat[0])
00062         lat = clim->hno3_lat[0];
00063     else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
00064         lat = clim->hno3_lat[clim->hno3_nlat - 1];
00065
00066     /* Get indices... */
```

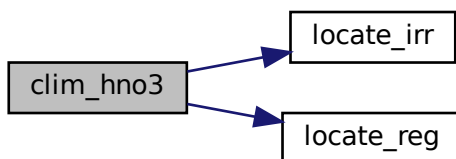


```

00067     int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
00068     int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
00069     int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);
00070
00071     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00072     double aux00 = LIN(clim->hno3_p[ip],
00073                       clim->hno3[isec][ilat][ip],
00074                       clim->hno3_p[ip + 1],
00075                       clim->hno3[isec][ilat][ip + 1], p);
00076     double aux01 = LIN(clim->hno3_p[ip],
00077                       clim->hno3[isec][ilat + 1][ip],
00078                       clim->hno3_p[ip + 1],
00079                       clim->hno3[isec][ilat + 1][ip + 1], p);
00080     double aux10 = LIN(clim->hno3_p[ip],
00081                       clim->hno3[isec + 1][ilat][ip],
00082                       clim->hno3_p[ip + 1],
00083                       clim->hno3[isec + 1][ilat][ip + 1], p);
00084     double aux11 = LIN(clim->hno3_p[ip],
00085                       clim->hno3[isec + 1][ilat + 1][ip],
00086                       clim->hno3_p[ip + 1],
00087                       clim->hno3[isec + 1][ilat + 1][ip + 1], p);
00088     aux00 = LIN(clim->hno3_lat[ilat], aux00,
00089               clim->hno3_lat[ilat + 1], aux01, lat);
00090     aux11 = LIN(clim->hno3_lat[ilat], aux10,
00091               clim->hno3_lat[ilat + 1], aux11, lat);
00092     aux00 = LIN(clim->hno3_time[isec], aux00,
00093               clim->hno3_time[isec + 1], aux11, sec);
00094
00095     /* Convert from ppb to ppv... */
00096     return GSL_MAX(1e-9 * aux00, 0.0);
00097 }

```

Here is the call graph for this function:



**5.21.3.5 clim\_hno3\_init()** void clim\_hno3\_init (  
     clim\_t \* clim )

Initialization function for HNO3 climatology.

Definition at line 101 of file libtrac.c.

```

00102     {
00103
00104     /* Write info... */
00105     LOG(1, "Initialize HNO3 data...");
00106
00107     clim->hno3_ntime = 12;
00108     double hno3_time[12] = {
00109         1209600.00, 3888000.00, 6393600.00,
00110         9072000.00, 11664000.00, 14342400.00,
00111         16934400.00, 19612800.00, 22291200.00,
00112         24883200.00, 27561600.00, 30153600.00
00113     };
00114     memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));
00115
00116     clim->hno3_nlat = 18;
00117     double hno3_lat[18] = {

```

```

00118     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00119     5, 15, 25, 35, 45, 55, 65, 75, 85
00120 };
00121 memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));
00122
00123 clim->hno3_np = 10;
00124 double hno3_p[10] = {
00125     4.64159, 6.81292, 10, 14.678, 21.5443,
00126     31.6228, 46.4159, 68.1292, 100, 146.78
00127 };
00128 memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));
00129
00130 double hno3[12][18][10] = {
00131     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00132      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00133      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00134      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00135      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00136      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00137      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00138      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00139      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00140      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00141      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00142      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00143      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00144      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00145      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00146      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00147      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00148      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00149     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00150      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00151      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00152      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00153      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00154      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00155      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00156      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00157      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00158      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00159      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00160      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00161      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00162      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00163      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00164      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00165      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00166      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00167     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00168      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00169      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00170      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00171      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00172      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00173      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00174      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00175      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00176      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00177      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00178      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00179      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00180      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00181      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00182      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00183      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00184      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00185     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00186      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00187      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00188      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00189      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00190      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00191      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00192      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00193      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00194      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00195      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00196      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00197      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00198      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00199      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00200      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00201      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00202      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00203     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00204      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57}},

```

```

00205     {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00206     {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00207     {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00208     {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00209     {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00210     {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00211     {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00212     {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00213     {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00214     {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00215     {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00216     {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00217     {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00218     {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00219     {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00220     {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00221     {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00222     {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00223     {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00224     {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00225     {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00226     {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00227     {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00228     {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00229     {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00230     {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00231     {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00232     {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00233     {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00234     {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00235     {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00236     {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00237     {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00238     {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},
00239     {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00240     {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00241     {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00242     {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00243     {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00244     {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00245     {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00246     {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00247     {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00248     {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00249     {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00250     {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00251     {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00252     {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00253     {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00254     {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00255     {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00256     {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62},
00257     {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00258     {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00259     {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00260     {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00261     {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00262     {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00263     {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00264     {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00265     {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00266     {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00267     {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00268     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00269     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00270     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00271     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00272     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00273     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00274     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55},
00275     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00276     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00277     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00278     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00279     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00280     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00281     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00282     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00283     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00284     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00285     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00286     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00287     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00288     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00289     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00290     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00291     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},

```

```

00292     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00293     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91}},
00294     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84}},
00295     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97}},
00296     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}},
00297     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11}},
00298     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616}},
00299     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21}},
00300     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968}},
00301     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105}},
00302     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146}},
00303     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178}},
00304     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14}},
00305     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353}},
00306     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802}},
00307     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}},
00308     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52}},
00309     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73}},
00310     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00311     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}},
00312     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}},
00313     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75}},
00314     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41}},
00315     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955}},
00316     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61}},
00317     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269}},
00318     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132}},
00319     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121}},
00320     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}},
00321     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146}},
00322     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}},
00323     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448}},
00324     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948}},
00325     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56}},
00326     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76}},
00327     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97}},
00328     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00329     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89}},
00330     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74}},
00331     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65}},
00332     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28}},
00333     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837}},
00334     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488}},
00335     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256}},
00336     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198}},
00337     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173}},
00338     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138}},
00339     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133}},
00340     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189}},
00341     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6}},
00342     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39}},
00343     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85}},
00344     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24}},
00345     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}},
00346     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00347 };
00348 memcpy(clim->hno3, hno3, sizeof(clim->hno3));
00349
00350 /* Get range... */
00351 double hno3min = 1e99, hno3max = -1e99;
00352 for (int it = 0; it < clim->hno3_ntime; it++)
00353     for (int iz = 0; iz < clim->hno3_np; iz++)
00354         for (int iy = 0; iy < clim->hno3_nlat; iy++) {
00355             hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
00356             hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
00357         }
00358
00359 /* Write info... */
00360 LOG(2, "Number of time steps: %d", clim->hno3_ntime);
00361 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00362     clim->hno3_time[0], clim->hno3_time[1],
00363     clim->hno3_time[clim->hno3_ntime - 1]);
00364 LOG(2, "Number of pressure levels: %d", clim->hno3_np);
00365 LOG(2, "Altitude levels: %g, %g ... %g km",
00366     Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
00367     Z(clim->hno3_p[clim->hno3_np - 1]));
00368 LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
00369     clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
00370 LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
00371 LOG(2, "Latitudes: %g, %g ... %g deg",
00372     clim->hno3_lat[0], clim->hno3_lat[1],
00373     clim->hno3_lat[clim->hno3_nlat - 1]);
00374 LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
00375     1e-9 * hno3max);
00376 }

```

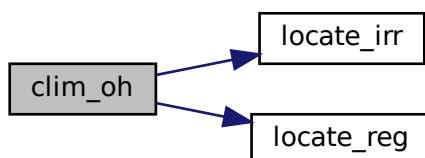
**5.21.3.6 clim\_oh()** double clim\_oh (  
     clim\_t \* clim,  
     double t,  
     double lat,  
     double p )

Climatology of OH number concentrations.

Definition at line 380 of file libtrac.c.

```
00384     {
00385
00386     /* Get seconds since begin of year... */
00387     double sec = FMOD(t, 365.25 * 86400.);
00388     while (sec < 0)
00389         sec += 365.25 * 86400.;
00390
00391     /* Check pressure... */
00392     if (p < clim->oh_p[clim->oh_np - 1])
00393         p = clim->oh_p[clim->oh_np - 1];
00394     else if (p > clim->oh_p[0])
00395         p = clim->oh_p[0];
00396
00397     /* Check latitude... */
00398     if (lat < clim->oh_lat[0])
00399         lat = clim->oh_lat[0];
00400     else if (lat > clim->oh_lat[clim->oh_nlat - 1])
00401         lat = clim->oh_lat[clim->oh_nlat - 1];
00402
00403     /* Get indices... */
00404     int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
00405     int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
00406     int ip = locate_irr(clim->oh_p, clim->oh_np, p);
00407
00408     /* Interpolate OH climatology... */
00409     double aux00 = LIN(clim->oh_p[ip],
00410                       clim->oh[isec][ip][ilat],
00411                       clim->oh_p[ip + 1],
00412                       clim->oh[isec][ip + 1][ilat], p);
00413     double aux01 = LIN(clim->oh_p[ip],
00414                       clim->oh[isec][ip][ilat + 1],
00415                       clim->oh_p[ip + 1],
00416                       clim->oh[isec][ip + 1][ilat + 1], p);
00417     double aux10 = LIN(clim->oh_p[ip],
00418                       clim->oh[isec + 1][ip][ilat],
00419                       clim->oh_p[ip + 1],
00420                       clim->oh[isec + 1][ip + 1][ilat], p);
00421     double aux11 = LIN(clim->oh_p[ip],
00422                       clim->oh[isec + 1][ip][ilat + 1],
00423                       clim->oh_p[ip + 1],
00424                       clim->oh[isec + 1][ip + 1][ilat + 1], p);
00425     aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
00426     aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
00427     aux00 =
00428         LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);
00429
00430     return GSL_MAX(aux00, 0.0);
00431 }
```

Here is the call graph for this function:



```

5.21.3.7 clim_oh_diurnal() double clim_oh_diurnal (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    double p,
    double lon,
    double lat )

```

Climatology of OH number concentrations with diurnal variation.

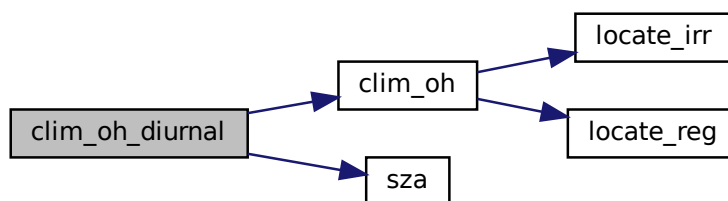
Definition at line 435 of file [libtrac.c](#).

```

00441     {
00442
00443     double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);
00444
00445     if (sza2 <= M_PI / 2. * 89. / 90.)
00446         return oh * exp(-ctl->oh_chem_beta / cos(sza2));
00447     else
00448         return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
00449 }

```

Here is the call graph for this function:



```

5.21.3.8 clim_oh_init() void clim_oh_init (
    ctl_t * ctl,
    clim_t * clim )

```

Initialization function for OH climatology.

Definition at line 453 of file [libtrac.c](#).

```

00455     {
00456
00457     int nt, ncid, varid;
00458
00459     double *help, ohmin = 1e99, ohmax = -1e99;
00460
00461     /* Write info... */
00462     LOG(1, "Read OH data: %s", ctl->clim_oh_filename);
00463
00464     /* Open netCDF file... */
00465     if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00466         WARN("OH climatology data are missing!");
00467         return;
00468     }
00469
00470     /* Read pressure data... */
00471     NC_INQ_DIM("press", &clim->oh_np, 2, CP);
00472     NC_GET_DOUBLE("press", clim->oh_p, 1);
00473
00474     /* Check ordering of pressure data... */

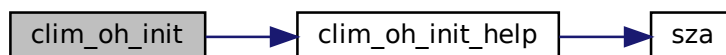
```

```

00475     if (clim->oh_p[0] < clim->oh_p[1])
00476         ERRMSG("Pressure data are not descending!");
00477
00478     /* Read latitudes... */
00479     NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
00480     NC_GET_DOUBLE("lat", clim->oh_lat, 1);
00481
00482     /* Check ordering of latitudes... */
00483     if (clim->oh_lat[0] > clim->oh_lat[1])
00484         ERRMSG("Latitude data are not ascending!");
00485
00486     /* Set time data for monthly means... */
00487     clim->oh_ntime = 12;
00488     clim->oh_time[0] = 1209600.00;
00489     clim->oh_time[1] = 3888000.00;
00490     clim->oh_time[2] = 6393600.00;
00491     clim->oh_time[3] = 9072000.00;
00492     clim->oh_time[4] = 11664000.00;
00493     clim->oh_time[5] = 14342400.00;
00494     clim->oh_time[6] = 16934400.00;
00495     clim->oh_time[7] = 19612800.00;
00496     clim->oh_time[8] = 22291200.00;
00497     clim->oh_time[9] = 24883200.00;
00498     clim->oh_time[10] = 27561600.00;
00499     clim->oh_time[11] = 30153600.00;
00500
00501     /* Check number of timesteps... */
00502     NC_INQ_DIM("time", &nt, 12, 12);
00503
00504     /* Read OH data... */
00505     ALLOC(help, double,
00506           clim->oh_nlat * clim->oh_np * clim->oh_ntime);
00507     NC_GET_DOUBLE("OH", help, 1);
00508     for (int it = 0; it < clim->oh_ntime; it++)
00509         for (int iz = 0; iz < clim->oh_np; iz++)
00510             for (int iy = 0; iy < clim->oh_nlat; iy++) {
00511                 clim->oh[it][iz][iy] =
00512                     help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
00513                     / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
00514                                         clim->oh_lat[iy]);
00515                 ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
00516                 ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
00517             }
00518     free(help);
00519
00520     /* Close netCDF file... */
00521     NC(nc_close(ncid));
00522
00523     /* Write info... */
00524     LOG(2, "Number of time steps: %d", clim->oh_ntime);
00525     LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00526         clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
00527     LOG(2, "Number of pressure levels: %d", clim->oh_np);
00528     LOG(2, "Altitude levels: %g, %g ... %g km",
00529         Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
00530     LOG(2, "Pressure levels: %g, %g ... %g hPa",
00531         clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
00532     LOG(2, "Number of latitudes: %d", clim->oh_nlat);
00533     LOG(2, "Latitudes: %g, %g ... %g deg",
00534         clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
00535     LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
00536 }

```

Here is the call graph for this function:



```

5.21.3.9 clim_oh_init_help() double clim_oh_init_help (
    double beta,
    double time,
    double lat )

```

Apply diurnal correction to OH climatology.

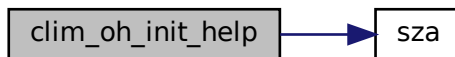
Definition at line 540 of file [libtrac.c](#).

```

00543     {
00544
00545     double aux, lon, sum = 0;
00546
00547     int n = 0;
00548
00549     /* Integrate day/night correction factor over longitude... */
00550     for (lon = -180; lon < 180; lon += 1) {
00551         aux = sza(time, lon, lat);
00552         if (aux <= M_PI / 2. * 85. / 90.)
00553             sum += exp(-beta / cos(aux));
00554         else
00555             sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
00556         n++;
00557     }
00558     return sum / (double) n;
00559 }

```

Here is the call graph for this function:



```

5.21.3.10 clim_h2o2() double clim_h2o2 (
    clim_t * clim,
    double t,
    double lat,
    double p )

```

Climatology of H2O2 number concentrations.

Definition at line 563 of file [libtrac.c](#).

```

00567     {
00568
00569     /* Get seconds since begin of year... */
00570     double sec = FMOD(t, 365.25 * 86400.);
00571     while (sec < 0)
00572         sec += 365.25 * 86400.;
00573
00574     /* Check pressure... */
00575     if (p < clim->h2o2_p[clim->h2o2_np - 1])
00576         p = clim->h2o2_p[clim->h2o2_np - 1];
00577     else if (p > clim->h2o2_p[0])
00578         p = clim->h2o2_p[0];
00579
00580     /* Check latitude... */
00581     if (lat < clim->h2o2_lat[0])
00582         lat = clim->h2o2_lat[0];
00583     else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
00584         lat = clim->h2o2_lat[clim->h2o2_nlat - 1];
00585 }

```

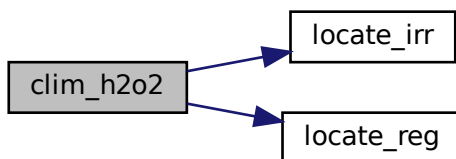


```

00586  /* Get indices... */
00587  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
00588  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
00589  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);
00590
00591  /* Interpolate H2O2 climatology... */
00592  double aux00 = LIN(clim->h2o2_p[ip],
00593                    clim->h2o2[isec][ip][ilat],
00594                    clim->h2o2_p[ip + 1],
00595                    clim->h2o2[isec][ip + 1][ilat], p);
00596  double aux01 = LIN(clim->h2o2_p[ip],
00597                    clim->h2o2[isec][ip][ilat + 1],
00598                    clim->h2o2_p[ip + 1],
00599                    clim->h2o2[isec][ip + 1][ilat + 1], p);
00600  double aux10 = LIN(clim->h2o2_p[ip],
00601                    clim->h2o2[isec + 1][ip][ilat],
00602                    clim->h2o2_p[ip + 1],
00603                    clim->h2o2[isec + 1][ip + 1][ilat], p);
00604  double aux11 = LIN(clim->h2o2_p[ip],
00605                    clim->h2o2[isec + 1][ip][ilat + 1],
00606                    clim->h2o2_p[ip + 1],
00607                    clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
00608  aux00 =
00609      LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
00610  aux11 =
00611      LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
00612  aux00 =
00613      LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);
00614
00615  return GSL_MAX(aux00, 0.0);
00616 }

```

Here is the call graph for this function:



**5.21.3.11 clim\_h2o2\_init()** void clim\_h2o2\_init (

```

    ctl_t * ctl,
    clim_t * clim )

```

Initialization function for H2O2 climatology.

Definition at line 620 of file [libtrac.c](#).

```

00622  {
00623
00624  int ncid, varid, it, iy, iz, nt;
00625
00626  double *help, h2o2min = 1e99, h2o2max = -1e99;
00627
00628  /* Write info... */
00629  LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);
00630
00631  /* Open netCDF file... */
00632  if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00633      WARN("H2O2 climatology data are missing!");
00634      return;
00635  }
00636

```

```

00637  /* Read pressure data... */
00638  NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
00639  NC_GET_DOUBLE("press", clim->h2o2_p, 1);
00640
00641  /* Check ordering of pressure data... */
00642  if (clim->h2o2_p[0] < clim->h2o2_p[1])
00643      ERRMSG("Pressure data are not descending!");
00644
00645  /* Read latitudes... */
00646  NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
00647  NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);
00648
00649  /* Check ordering of latitude data... */
00650  if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
00651      ERRMSG("Latitude data are not ascending!");
00652
00653  /* Set time data (for monthly means)... */
00654  clim->h2o2_ntime = 12;
00655  clim->h2o2_time[0] = 1209600.00;
00656  clim->h2o2_time[1] = 3888000.00;
00657  clim->h2o2_time[2] = 6393600.00;
00658  clim->h2o2_time[3] = 9072000.00;
00659  clim->h2o2_time[4] = 11664000.00;
00660  clim->h2o2_time[5] = 14342400.00;
00661  clim->h2o2_time[6] = 16934400.00;
00662  clim->h2o2_time[7] = 19612800.00;
00663  clim->h2o2_time[8] = 22291200.00;
00664  clim->h2o2_time[9] = 24883200.00;
00665  clim->h2o2_time[10] = 27561600.00;
00666  clim->h2o2_time[11] = 30153600.00;
00667
00668  /* Check number of timesteps... */
00669  NC_INQ_DIM("time", &nt, 12, 12);
00670
00671  /* Read data... */
00672  ALLOC(help, double,
00673         clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
00674  NC_GET_DOUBLE("h2o2", help, 1);
00675  for (it = 0; it < clim->h2o2_ntime; it++)
00676      for (iz = 0; iz < clim->h2o2_np; iz++)
00677          for (iy = 0; iy < clim->h2o2_nlat; iy++) {
00678              clim->h2o2[it][iz][iy] =
00679                  help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
00680              h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
00681              h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
00682          }
00683  free(help);
00684
00685  /* Close netCDF file... */
00686  NC(nc_close(ncid));
00687
00688  /* Write info... */
00689  LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
00690  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00691       clim->h2o2_time[0], clim->h2o2_time[1],
00692       clim->h2o2_time[clim->h2o2_ntime - 1]);
00693  LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
00694  LOG(2, "Altitude levels: %g, %g ... %g km",
00695       Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
00696       Z(clim->h2o2_p[clim->h2o2_np - 1]));
00697  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
00698       clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
00699  LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
00700  LOG(2, "Latitudes: %g, %g ... %g deg",
00701       clim->h2o2_lat[0], clim->h2o2_lat[1],
00702       clim->h2o2_lat[clim->h2o2_nlat - 1]);
00703  LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
00704 }

```

```

5.21.3.12 clim_tropo() double clim_tropo (
    clim_t * clim,
    double t,
    double lat )

```

Climatology of tropopause pressure.

Definition at line 708 of file libtrac.c.

```

00711 {

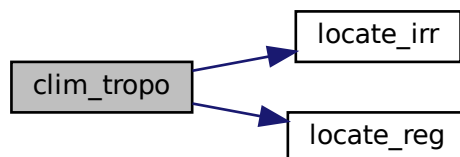
```

```

00712
00713  /* Get seconds since begin of year... */
00714  double sec = FMOD(t, 365.25 * 86400.);
00715  while (sec < 0)
00716      sec += 365.25 * 86400.;
00717
00718  /* Get indices... */
00719  int isec = locate_irr(clim->tropo_time, clim->tropo_ntime, sec);
00720  int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);
00721
00722  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
00723  double p0 = LIN(clim->tropo_lat[ilat],
00724                 clim->tropo[isec][ilat],
00725                 clim->tropo_lat[ilat + 1],
00726                 clim->tropo[isec][ilat + 1], lat);
00727  double p1 = LIN(clim->tropo_lat[ilat],
00728                 clim->tropo[isec + 1][ilat],
00729                 clim->tropo_lat[ilat + 1],
00730                 clim->tropo[isec + 1][ilat + 1], lat);
00731  return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
00732 }

```

Here is the call graph for this function:



**5.21.3.13 clim\_tropo\_init()** void clim\_tropo\_init (  
     clim\_t \* clim )

Initialize tropopause climatology.

Definition at line 736 of file libtrac.c.

```

00737  {
00738
00739  /* Write info... */
00740  LOG(1, "Initialize tropopause data...");
00741
00742  clim->tropo_ntime = 12;
00743  double tropo_time[12] = {
00744      1209600.00, 3888000.00, 6393600.00,
00745      9072000.00, 11664000.00, 14342400.00,
00746      16934400.00, 19612800.00, 22291200.00,
00747      24883200.00, 27561600.00, 30153600.00
00748  };
00749  memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));
00750
00751  clim->tropo_nlat = 73;
00752  double tropo_lat[73] = {
00753      -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00754      -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00755      -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00756      -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00757      15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00758      45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00759      75, 77.5, 80, 82.5, 85, 87.5, 90
00760  };
00761  memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));
00762

```

```
00763 double tropo[12][73] = {
00764 {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00765 297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00766 175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00767 99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00768 98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00769 152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00770 277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00771 275.3, 275.6, 275.4, 274.1, 273.5},
00772 {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00773 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00774 150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00775 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00776 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00777 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00778 284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00779 287.5, 286.2, 285.8},
00780 {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00781 297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00782 161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00783 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00784 99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00785 186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00786 279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00787 304.3, 304.9, 306, 306.6, 306.2, 306},
00788 {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00789 290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00790 195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00791 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00792 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00793 148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00794 263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00795 315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00796 {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00797 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00798 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00799 101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00800 102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00801 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00802 273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00803 325.3, 325.8, 325.8},
00804 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00805 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00806 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00807 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00808 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00809 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00810 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00811 308.5, 312.2, 313.1, 313.3},
00812 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00813 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00814 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00815 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00816 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00817 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00818 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00819 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00820 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00821 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00822 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00823 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00824 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00825 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00826 230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00827 278.2, 282.6, 287.4, 290.9, 292.5, 293},
00828 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00829 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00830 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00831 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00832 110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00833 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00834 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00835 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00836 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00837 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00838 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00839 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00840 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00841 112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00842 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00843 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00844 305.1},
00845 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00846 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00847 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00848 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00849 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
```

```

00850     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00851     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00852     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00853     {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00854     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00855     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00856     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00857     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00858     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00859     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00860     281.7, 281.1, 281.2}
00861 };
00862 memcpy(clim->tropo, tropo, sizeof(clim->tropo));
00863
00864 /* Get range... */
00865 double tropomin = 1e99, tropomax = -1e99;
00866 for (int it = 0; it < clim->tropo_ntime; it++)
00867     for (int iy = 0; iy < clim->tropo_nlat; iy++) {
00868         tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
00869         tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
00870     }
00871
00872 /* Write info... */
00873 LOG(2, "Number of time steps: %d", clim->tropo_ntime);
00874 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00875     clim->tropo_time[0], clim->tropo_time[1],
00876     clim->tropo_time[clim->tropo_ntime - 1]);
00877 LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
00878 LOG(2, "Latitudes: %g, %g ... %g deg",
00879     clim->tropo_lat[0], clim->tropo_lat[1],
00880     clim->tropo_lat[clim->tropo_nlat - 1]);
00881 LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
00882     Z(tropomin));
00883 LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
00884 }

```

**5.21.3.14 compress\_pack()** void compress\_pack (

```

    char * varname,
    float * array,
    size_t nxy,
    size_t nz,
    int decompress,
    FILE * inout )

```

Pack or unpack array.

Definition at line 888 of file libtrac.c.

```

00894     {
00895
00896     double min[EP], max[EP], off[EP], scl[EP];
00897
00898     unsigned short *sarray;
00899
00900     /* Allocate... */
00901     ALLOC(sarray, unsigned short,
00902         nxy * nz);
00903
00904     /* Read compressed stream and decompress array... */
00905     if (decompress) {
00906
00907         /* Write info... */
00908         LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
00909             varname, 100. * sizeof(unsigned short) / sizeof(float));
00910
00911         /* Read data... */
00912         FREAD(&scl, double,
00913             nz,
00914             inout);
00915         FREAD(&off, double,
00916             nz,
00917             inout);
00918         FREAD(sarray, unsigned short,
00919             nxy * nz,
00920             inout);
00921
00922         /* Convert to float... */

```

```

00923 #pragma omp parallel for default(shared)
00924     for (size_t ixy = 0; ixy < nxy; ixy++)
00925         for (size_t iz = 0; iz < nz; iz++)
00926             array[ixy * nz + iz]
00927                 = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
00928     }
00929
00930     /* Compress array and output compressed stream... */
00931     else {
00932
00933         /* Write info... */
00934         LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
00935             varname, 100. * sizeof(unsigned short) / sizeof(float));
00936
00937         /* Get range... */
00938         for (size_t iz = 0; iz < nz; iz++) {
00939             min[iz] = array[iz];
00940             max[iz] = array[iz];
00941         }
00942         for (size_t ixy = 1; ixy < nxy; ixy++)
00943             for (size_t iz = 0; iz < nz; iz++) {
00944                 if (array[ixy * nz + iz] < min[iz])
00945                     min[iz] = array[ixy * nz + iz];
00946                 if (array[ixy * nz + iz] > max[iz])
00947                     max[iz] = array[ixy * nz + iz];
00948             }
00949
00950         /* Get offset and scaling factor... */
00951         for (size_t iz = 0; iz < nz; iz++) {
00952             scl[iz] = (max[iz] - min[iz]) / 65533.;
00953             off[iz] = min[iz];
00954         }
00955
00956         /* Convert to short... */
00957         #pragma omp parallel for default(shared)
00958         for (size_t ixy = 0; ixy < nxy; ixy++)
00959             for (size_t iz = 0; iz < nz; iz++)
00960                 if (scl[iz] != 0)
00961                     sarray[ixy * nz + iz] = (unsigned short)
00962                         ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
00963                 else
00964                     sarray[ixy * nz + iz] = 0;
00965
00966         /* Write data... */
00967         FWRITE(&scl, double,
00968             nz,
00969             inout);
00970         FWRITE(&off, double,
00971             nz,
00972             inout);
00973         FWRITE(sarray, unsigned short,
00974             nxy * nz,
00975             inout);
00976     }
00977
00978     /* Free... */
00979     free(sarray);
00980 }

```

**5.21.3.15 day2doy()** void day2doy (

```

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

```

Compress or decompress array with zfp.

Compress or decompress array with zstd.

Get day of year from date.

Definition at line 1129 of file libtrac.c.

```

01133     {
01134
01135     const int
01136         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },

```

```

01137     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01138
01139     /* Get day of year... */
01140     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01141         *doy = d0l[mon - 1] + day - 1;
01142     else
01143         *doy = d0[mon - 1] + day - 1;
01144 }

```

**5.21.3.16 doym2day()** void doym2day (

```

    int year,
    int doym,
    int * mon,
    int * day )

```

Get date from day of year.

Definition at line 1148 of file libtrac.c.

```

01152     {
01153
01154     const int
01155     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01156     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01157
01158     int i;
01159
01160     /* Get month and day... */
01161     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01162         for (i = 11; i > 0; i--)
01163             if (d0l[i] <= doym)
01164                 break;
01165         *mon = i + 1;
01166         *day = doym - d0l[i] + 1;
01167     } else {
01168         for (i = 11; i > 0; i--)
01169             if (d0[i] <= doym)
01170                 break;
01171         *mon = i + 1;
01172         *day = doym - d0[i] + 1;
01173     }
01174 }

```

**5.21.3.17 geo2cart()** void geo2cart (

```

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

```

Convert geolocation to Cartesian coordinates.

Definition at line 1178 of file libtrac.c.

```

01182     {
01183
01184     double radius = z + RE;
01185     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01186     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01187     x[2] = radius * sin(lat / 180. * M_PI);
01188 }

```

```

5.21.3.18 get_met() void get_met (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    met_t ** met0,
    met_t ** met1 )

```

Get meteo data for given time step.

Definition at line 1192 of file libtrac.c.

```

01197     {
01198
01199     static int init;
01200
01201     met_t *mets;
01202
01203     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01204
01205     /* Set timer... */
01206     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01207
01208     /* Init... */
01209     if (t == ctl->t_start || !init) {
01210         init = 1;
01211
01212         /* Read meteo data... */
01213         get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
01214                     ctl->metbase, ctl->dt_met, filename);
01215         if (!read_met(filename, ctl, clim, *met0))
01216             ERRMSG("Cannot open file!");
01217
01218         get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
01219                     ctl->metbase, ctl->dt_met, filename);
01220         if (!read_met(filename, ctl, clim, *met1))
01221             ERRMSG("Cannot open file!");
01222
01223         /* Update GPU... */
01224 #ifdef _OPENACC
01225         met_t *met0up = *met0;
01226         met_t *met1up = *met1;
01227 #ifdef ASYNCIO
01228 #pragma acc update device(met0up[:1],met1up[:1]) async(5)
01229 #else
01230 #pragma acc update device(met0up[:1],met1up[:1])
01231 #endif
01232 #endif
01233
01234         /* Caching... */
01235         if (ctl->met_cache && t != ctl->t_stop) {
01236             get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
01237                         ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
01238             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01239             LOG(1, "Caching: %s", cachefile);
01240             if (system(cmd) != 0)
01241                 WARN("Caching command failed!");
01242         }
01243     }
01244
01245     /* Read new data for forward trajectories... */
01246     if (t > (*met1->time) {
01247
01248         /* Pointer swap... */
01249         mets = *met1;
01250         *met1 = *met0;
01251         *met0 = mets;
01252
01253         /* Read new meteo data... */
01254         get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
01255         if (!read_met(filename, ctl, clim, *met1))
01256             ERRMSG("Cannot open file!");
01257         /* Update GPU... */
01258 #ifdef _OPENACC
01259         met_t *met1up = *met1;
01260 #ifdef ASYNCIO
01261 #pragma acc update device(met1up[:1]) async(5)
01262 #else
01263 #pragma acc update device(met1up[:1])
01264 #endif
01265 #endif
01266         /* Caching... */
01267         if (ctl->met_cache && t != ctl->t_stop) {
01268             get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
01269                         cachefile);

```

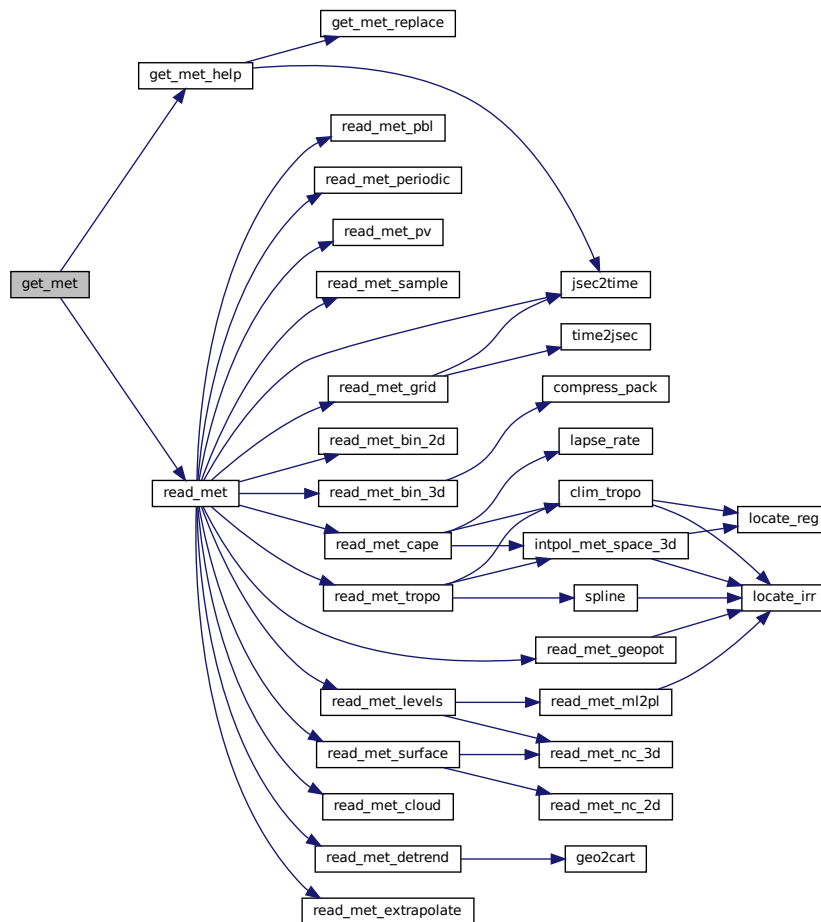


```

01270     sprintf(cmd, "cat %s > /dev/null &", cachefile);
01271     LOG(1, "Caching: %s", cachefile);
01272     if (system(cmd) != 0)
01273         WARN("Caching command failed!");
01274     }
01275 }
01276 /* Read new data for backward trajectories... */
01277 if (t < (*met0)->time) {
01278
01279     /* Pointer swap... */
01280     mets = *met1;
01281     *met1 = *met0;
01282     *met0 = mets;
01283
01284     /* Read new meteo data... */
01285     get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
01286     if (!read_met(filename, ctl, clim, *met0))
01287         ERRMSG("Cannot open file!");
01288
01289     /* Update GPU... */
01290 #ifdef _OPENACC
01291     met_t *met0up = *met0;
01292 #ifdef ASYNCIO
01293     #pragma acc update device(met0up[:1]) async(5)
01294 #else
01295     #pragma acc update device(met0up[:1])
01296 #endif
01297 #endif
01298
01299     /* Caching... */
01300     if (ctl->met_cache && t != ctl->t_stop) {
01301         get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
01302             cachefile);
01303         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01304         LOG(1, "Caching: %s", cachefile);
01305         if (system(cmd) != 0)
01306             WARN("Caching command failed!");
01307     }
01308 }
01309 /* Check that grids are consistent... */
01310 if ((*met0)->nx != 0 && (*met1)->nx != 0) {
01311     if ((*met0)->nx != (*met1)->nx
01312         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01313         ERRMSG("Meteo grid dimensions do not match!");
01314     for (int ix = 0; ix < (*met0)->nx; ix++)
01315         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01316             ERRMSG("Meteo grid longitudes do not match!");
01317     for (int iy = 0; iy < (*met0)->ny; iy++)
01318         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01319             ERRMSG("Meteo grid latitudes do not match!");
01320     for (int ip = 0; ip < (*met0)->np; ip++)
01321         if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01322             ERRMSG("Meteo grid pressure levels do not match!");
01323 }
01324 }

```

Here is the call graph for this function:



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

```

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

```

Get meteo data for time step.

Definition at line 1328 of file libtrac.c.

```

01334     {
01335
01336     char repl[LEN];
01337
01338     double t6, r;
01339
01340     int year, mon, day, hour, min, sec;
01341
01342     /* Round time to fixed intervals... */
01343     if (direct == -1)
01344         t6 = floor(t / dt_met) * dt_met;

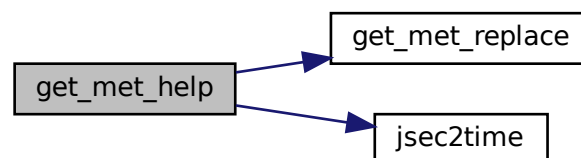
```

```

01345     else
01346         t6 = ceil(t / dt_met) * dt_met;
01347
01348     /* Decode time... */
01349     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01350
01351     /* Set filename of MPTRAC meteo files... */
01352     if (ctl->clams_met_data == 0) {
01353         if (ctl->met_type == 0)
01354             sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01355         else if (ctl->met_type == 1)
01356             sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
01357         else if (ctl->met_type == 2)
01358             sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
01359         else if (ctl->met_type == 3)
01360             sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
01361         else if (ctl->met_type == 4)
01362             sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", metbase);
01363         sprintf(repl, "%d", year);
01364         get_met_replace(filename, "YYYY", repl);
01365         sprintf(repl, "%02d", mon);
01366         get_met_replace(filename, "MM", repl);
01367         sprintf(repl, "%02d", day);
01368         get_met_replace(filename, "DD", repl);
01369         sprintf(repl, "%02d", hour);
01370         get_met_replace(filename, "HH", repl);
01371     }
01372
01373     /* Set filename of CLaMS meteo files... */
01374     else {
01375         sprintf(filename, "%s_YYMMDDHH.nc", metbase);
01376         sprintf(repl, "%d", year);
01377         get_met_replace(filename, "YYYY", repl);
01378         sprintf(repl, "%d", year % 100);
01379         get_met_replace(filename, "YY", repl);
01380         sprintf(repl, "%02d", mon);
01381         get_met_replace(filename, "MM", repl);
01382         sprintf(repl, "%02d", day);
01383         get_met_replace(filename, "DD", repl);
01384         sprintf(repl, "%02d", hour);
01385         get_met_replace(filename, "HH", repl);
01386     }
01387 }

```

Here is the call graph for this function:



**5.21.3.20 get\_met\_replace()** void get\_met\_replace (  
char \* orig,  
char \* search,  
char \* repl )

Replace template strings in filename.

Definition at line 1391 of file libtrac.c.

```
01394 {
```

```

01395
01396 char buffer[LEN];
01397
01398 /* Iterate... */
01399 for (int i = 0; i < 3; i++) {
01400
01401     /* Replace sub-string... */
01402     char *ch;
01403     if (!(ch = strstr(orig, search)))
01404         return;
01405     strncpy(buffer, orig, (size_t) (ch - orig));
01406     buffer[ch - orig] = 0;
01407     sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01408     orig[0] = 0;
01409     strcpy(orig, buffer);
01410 }
01411 }

```

### 5.21.3.21 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 meteo data.

Definition at line 1415 of file libtrac.c.

```

01424     {
01425
01426     /* Initialize interpolation... */
01427     if (init) {
01428
01429         /* Check longitude... */
01430         if (met->lon[met->nx - 1] > 180 && lon < 0)
01431             lon += 360;
01432
01433         /* Get interpolation indices... */
01434         ci[0] = locate_irr(met->p, met->np, p);
01435         ci[1] = locate_reg(met->lon, met->nx, lon);
01436         ci[2] = locate_reg(met->lat, met->ny, lat);
01437
01438         /* Get interpolation weights... */
01439         cw[0] = (met->p[ci[0] + 1] - p)
01440             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01441         cw[1] = (met->lon[ci[1] + 1] - lon)
01442             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01443         cw[2] = (met->lat[ci[2] + 1] - lat)
01444             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01445     }
01446
01447     /* Interpolate vertically... */
01448     double aux00 =
01449         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01450         + array[ci[1]][ci[2]][ci[0] + 1];
01451     double aux01 =
01452         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01453             array[ci[1]][ci[2] + 1][ci[0] + 1])
01454         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01455     double aux10 =
01456         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01457             array[ci[1] + 1][ci[2]][ci[0] + 1])
01458         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01459     double aux11 =
01460         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01461             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01462         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01463
01464     /* Interpolate horizontally... */
01465     aux00 = cw[2] * (aux00 - aux01) + aux01;
01466     aux11 = cw[2] * (aux10 - aux11) + aux11;

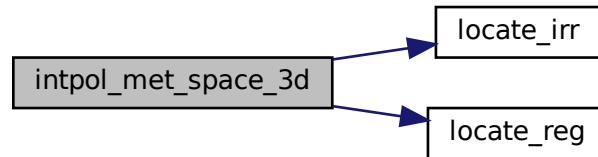
```

```

01467     *var = cw[1] * (aux00 - aux11) + aux11;
01468 }

```

Here is the call graph for this function:



### 5.21.3.22 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 meteo data.

Definition at line 1472 of file libtrac.c.

```

01480     {
01481
01482     /* Initialize interpolation... */
01483     if (init) {
01484
01485     /* Check longitude... */
01486     if (met->lon[met->nx - 1] > 180 && lon < 0)
01487         lon += 360;
01488
01489     /* Get interpolation indices... */
01490     ci[1] = locate_reg(met->lon, met->nx, lon);
01491     ci[2] = locate_reg(met->lat, met->ny, lat);
01492
01493     /* Get interpolation weights... */
01494     cw[1] = (met->lon[ci[1] + 1] - lon)
01495         / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01496     cw[2] = (met->lat[ci[2] + 1] - lat)
01497         / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01498     }
01499
01500     /* Set variables... */
01501     double aux00 = array[ci[1]][ci[2]];
01502     double aux01 = array[ci[1]][ci[2] + 1];
01503     double aux10 = array[ci[1] + 1][ci[2]];
01504     double aux11 = array[ci[1] + 1][ci[2] + 1];
01505
01506     /* Interpolate horizontally... */
01507     if (isfinite(aux00) && isfinite(aux01)
01508         && isfinite(aux10) && isfinite(aux11)) {
01509         aux00 = cw[2] * (aux00 - aux01) + aux01;
01510         aux11 = cw[2] * (aux10 - aux11) + aux11;
01511         *var = cw[1] * (aux00 - aux11) + aux11;
01512     } else {
01513         if (cw[2] < 0.5) {
01514             if (cw[1] < 0.5)

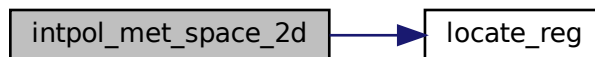
```

```

01515     *var = aux11;
01516     else
01517     *var = aux01;
01518 } else {
01519     if (cw[1] < 0.5)
01520     *var = aux10;
01521     else
01522     *var = aux00;
01523 }
01524 }
01525 }

```

Here is the call graph for this function:



**5.21.3.23 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 )

```

Spatial interpolation of meteo data.

Temporal interpolation of meteo data.

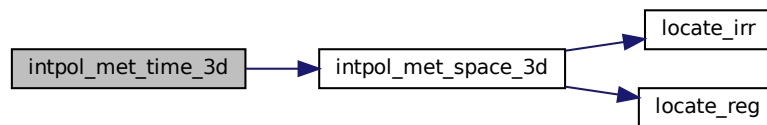
Definition at line 1632 of file libtrac.c.

```

01644     {
01645
01646     double var0, var1, wt;
01647
01648     /* Spatial interpolation... */
01649     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01650     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01651
01652     /* Get weighting factor... */
01653     wt = (met1->time - ts) / (met1->time - met0->time);
01654
01655     /* Interpolate... */
01656     *var = wt * (var0 - var1) + var1;
01657 }

```

Here is the call graph for this function:



### 5.21.3.24 `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 meteo data.

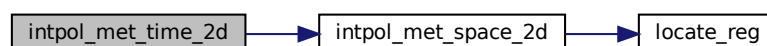
Definition at line 1661 of file [libtrac.c](#).

```

01672     {
01673
01674         double var0, var1, wt;
01675
01676         /* Spatial interpolation... */
01677         intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01678         intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01679
01680         /* Get weighting factor... */
01681         wt = (met1->time - ts) / (met1->time - met0->time);
01682
01683         /* Interpolate... */
01684         if (isfinite(var0) && isfinite(var1))
01685             *var = wt * (var0 - var1) + var1;
01686         else if (wt < 0.5)
01687             *var = var1;
01688         else
01689             *var = var0;
01690     }

```

Here is the call graph for this function:



```

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

```

Temporal interpolation of meteo data.

Convert seconds to date.

Definition at line 1725 of file libtrac.c.

```

01733     {
01734
01735     struct tm t0, *t1;
01736
01737     t0.tm_year = 100;
01738     t0.tm_mon = 0;
01739     t0.tm_mday = 1;
01740     t0.tm_hour = 0;
01741     t0.tm_min = 0;
01742     t0.tm_sec = 0;
01743
01744     time_t jsec0 = (time_t) jsec + timegm(&t0);
01745     t1 = gmtime(&jsec0);
01746
01747     *year = t1->tm_year + 1900;
01748     *mon = t1->tm_mon + 1;
01749     *day = t1->tm_mday;
01750     *hour = t1->tm_hour;
01751     *min = t1->tm_min;
01752     *sec = t1->tm_sec;
01753     *remain = jsec - floor(jsec);
01754 }

```

```

5.21.3.26 lapse_rate() double lapse_rate (
    double t,
    double h2o )

```

Calculate moist adiabatic lapse rate.

Definition at line 1758 of file libtrac.c.

```

01760     {
01761
01762     /*
01763      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01764      and water vapor volume mixing ratio [1].
01765
01766      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01767     */
01768
01769     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01770
01771     return 1e3 * GO * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01772 }

```



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

Find array index for irregular grid.

Definition at line 1776 of file [libtrac.c](#).

```
01779     {
01780
01781     int ilo = 0;
01782     int ihi = n - 1;
01783     int i = (ihi + ilo) » 1;
01784
01785     if (xx[i] < xx[i + 1])
01786         while (ihi > ilo + 1) {
01787             i = (ihi + ilo) » 1;
01788             if (xx[i] > x)
01789                 ihi = i;
01790             else
01791                 ilo = i;
01792         } else
01793             while (ihi > ilo + 1) {
01794                 i = (ihi + ilo) » 1;
01795                 if (xx[i] <= x)
01796                     ihi = i;
01797                 else
01798                     ilo = i;
01799             }
01800
01801     return ilo;
01802 }
```

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

Find array index for regular grid.

Definition at line 1806 of file [libtrac.c](#).

```
01809     {
01810
01811     /* Calculate index... */
01812     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01813
01814     /* Check range... */
01815     if (i < 0)
01816         return 0;
01817     else if (i > n - 2)
01818         return n - 2;
01819     else
01820         return i;
01821 }
```

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

Calculate NAT existence temperature.

Definition at line 1825 of file [libtrac.c](#).

```
01828     {
01829
01830     /* Check water vapor vmr... */
```

```

01831 h2o = GSL_MAX(h2o, 0.1e-6);
01832
01833 /* Calculate T_NAT... */
01834 double p_hno3 = hno3 * p / 1.333224;
01835 double p_h2o = h2o * p / 1.333224;
01836 double a = 0.009179 - 0.00088 * log10(p_h2o);
01837 double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01838 double c = -11397.0 / a;
01839 double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
01840 double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01841 if (x2 > 0)
01842     tnat = x2;
01843
01844 return tnat;
01845 }

```

**5.21.3.30 quicksort()** void quicksort (

```

    double arr[],
    int brr[],
    int low,
    int high )

```

Parallel quicksort.

Definition at line 1849 of file libtrac.c.

```

01853 {
01854
01855     if (low < high) {
01856         int pi = quicksort_partition(arr, brr, low, high);
01857
01858         #pragma omp task firstprivate(arr,brr,low,pi)
01859         {
01860             quicksort(arr, brr, low, pi - 1);
01861         }
01862
01863         // #pragma omp task firstprivate(arr,brr,high,pi)
01864         {
01865             quicksort(arr, brr, pi + 1, high);
01866         }
01867     }
01868 }

```

Here is the call graph for this function:



**5.21.3.31 quicksort\_partition()** int quicksort\_partition (

```

    double arr[],
    int brr[],
    int low,
    int high )

```

Partition function for quicksort.

Definition at line 1872 of file libtrac.c.

```

01876     {
01877
01878     double pivot = arr[high];
01879     int i = (low - 1);
01880
01881     for (int j = low; j <= high - 1; j++)
01882         if (arr[j] <= pivot) {
01883             i++;
01884             SWAP(arr[i], arr[j], double);
01885             SWAP(brr[i], brr[j], int);
01886         }
01887     SWAP(arr[high], arr[i + 1], double);
01888     SWAP(brr[high], brr[i + 1], int);
01889
01890     return (i + 1);
01891 }
```

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

Read atmospheric data.

Definition at line 1895 of file libtrac.c.

```

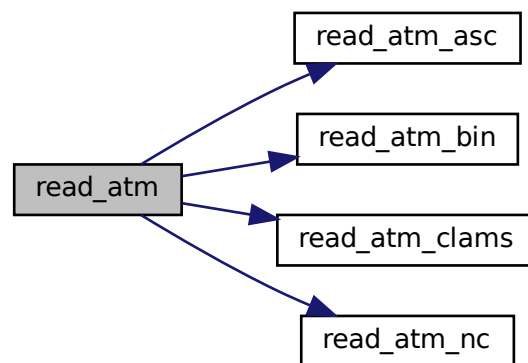
01898     {
01899
01900     int result;
01901
01902     /* Set timer... */
01903     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
01904
01905     /* Init... */
01906     atm->np = 0;
01907
01908     /* Write info... */
01909     LOG(1, "Read atmospheric data: %s", filename);
01910
01911     /* Read ASCII data... */
01912     if (ctl->atm_type == 0)
01913         result = read_atm_asc(filename, ctl, atm);
01914
01915     /* Read binary data... */
01916     else if (ctl->atm_type == 1)
01917         result = read_atm_bin(filename, ctl, atm);
01918
01919     /* Read netCDF data... */
01920     else if (ctl->atm_type == 2)
01921         result = read_atm_nc(filename, ctl, atm);
01922
01923     /* Read CLaMS data... */
01924     else if (ctl->atm_type == 3)
01925         result = read_atm_clams(filename, ctl, atm);
01926
01927     /* Error... */
01928     else
01929         ERRMSG("Atmospheric data type not supported!");
01930
01931     /* Check result... */
01932     if (result != 1)
01933         return 0;
01934
01935     /* Check number of air parcels... */
01936     if (atm->np < 1)
01937         ERRMSG("Can not read any data!");
01938
01939     /* Write info... */
01940     double mini, maxi;
01941     LOG(2, "Number of particles: %d", atm->np);
01942     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
01943     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
01944     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
01945     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
01946     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
01947     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
01948     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
01949     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
```

```

01950 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
01951 for (int iq = 0; iq < ctl->nq; iq++) {
01952     char msg[LEN];
01953     sprintf(msg, "Quantity %s range: %s ... %s %s",
01954             ctl->qnt_name[iq], ctl->qnt_format[iq],
01955             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
01956     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
01957     LOG(2, msg, mini, maxi);
01958 }
01959
01960 /* Return success... */
01961 return 1;
01962 }

```

Here is the call graph for this function:



**5.21.3.33 read\_atm\_asc()** `int read_atm_asc (`  
`const char * filename,`  
`ctl_t * ctl,`  
`atm_t * atm )`

Read atmospheric data in ASCII format.

Definition at line 1966 of file `libtrac.c`.

```

01969 {
01970
01971     FILE *in;
01972
01973     /* Open file... */
01974     if (!(in = fopen(filename, "r"))) {
01975         WARN("Cannot open file!");
01976         return 0;
01977     }
01978
01979     /* Read line... */
01980     char line[LEN];
01981     while (fgets(line, LEN, in)) {
01982
01983         /* Read data... */
01984         char *tok;
01985         TOK(line, tok, "%lg", atm->time[atm->np]);
01986         TOK(NULL, tok, "%lg", atm->p[atm->np]);
01987         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
01988         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
01989         for (int iq = 0; iq < ctl->nq; iq++)
01990             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

```

```

01991
01992     /* Convert altitude to pressure... */
01993     atm->p[atm->np] = P(atm->p[atm->np]);
01994
01995     /* Increment data point counter... */
01996     if ((++atm->np) > NP)
01997         ERRMSG("Too many data points!");
01998 }
01999
02000 /* Close file... */
02001 fclose(in);
02002
02003 /* Return success... */
02004 return 1;
02005 }

```

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

Read atmospheric data in binary format.

Definition at line 2009 of file libtrac.c.

```

02012     {
02013
02014     FILE *in;
02015
02016     /* Open file... */
02017     if (!(in = fopen(filename, "r")))
02018         return 0;
02019
02020     /* Check version of binary data... */
02021     int version;
02022     FREAD(&version, int,
02023          1,
02024          in);
02025     if (version != 100)
02026         ERRMSG("Wrong version of binary data!");
02027
02028     /* Read data... */
02029     FREAD(&atm->np, int,
02030          1,
02031          in);
02032     FREAD(atm->time, double,
02033          (size_t) atm->np,
02034          in);
02035     FREAD(atm->p, double,
02036          (size_t) atm->np,
02037          in);
02038     FREAD(atm->lon, double,
02039          (size_t) atm->np,
02040          in);
02041     FREAD(atm->lat, double,
02042          (size_t) atm->np,
02043          in);
02044     for (int iq = 0; iq < ctl->nq; iq++)
02045         FREAD(atm->q[iq], double,
02046              (size_t) atm->np,
02047              in);
02048
02049     /* Read final flag... */
02050     int final;
02051     FREAD(&final, int,
02052          1,
02053          in);
02054     if (final != 999)
02055         ERRMSG("Error while reading binary data!");
02056
02057     /* Close file... */
02058     fclose(in);
02059
02060     /* Return success... */
02061     return 1;
02062 }

```

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

Read atmospheric data in CLaMS format.

Definition at line 2066 of file libtrac.c.

```

02069     {
02070
02071     int ncid, varid;
02072
02073     /* Open file... */
02074     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02075         return 0;
02076
02077     /* Get dimensions... */
02078     NC_INQ_DIM("NPARTS", &atm->np, 1, NP);
02079
02080     /* Get time... */
02081     if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
02082         NC(nc_get_var_double(ncid, varid, atm->time));
02083     } else {
02084         WARN("TIME_INIT not found use time instead!");
02085         double time_init;
02086         NC_GET_DOUBLE("time", &time_init, 1);
02087         for (int ip = 0; ip < atm->np; ip++) {
02088             atm->time[ip] = time_init;
02089         }
02090     }
02091
02092     /* Read zeta coordinate, pressure is optional... */
02093     if (ctl->vert_coord_ap == 1) {
02094         NC_GET_DOUBLE("ZETA", atm->zeta, 1);
02095         NC_GET_DOUBLE("PRESS", atm->p, 0);
02096     }
02097
02098     /* Read pressure, zeta coordinate is optional... */
02099     else {
02100         NC_GET_DOUBLE("PRESS", atm->p, 1);
02101         NC_GET_DOUBLE("ZETA", atm->zeta, 0);
02102     }
02103
02104     /* Read longitude and latitude... */
02105     NC_GET_DOUBLE("LON", atm->lon, 1);
02106     NC_GET_DOUBLE("LAT", atm->lat, 1);
02107
02108     /* Close file... */
02109     NC(nc_close(ncid));
02110
02111     /* Return success... */
02112     return 1;
02113 }

```

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

Read atmospheric data in netCDF format.

Definition at line 2117 of file libtrac.c.

```

02120     {
02121
02122     int ncid, varid;
02123
02124     /* Open file... */
02125     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02126         return 0;
02127
02128     /* Get dimensions... */
02129     NC_INQ_DIM("obs", &atm->np, 1, NP);
02130
02131     /* Read geolocations... */
02132     NC_GET_DOUBLE("time", atm->time, 1);

```

```

02133 NC_GET_DOUBLE("press", atm->p, 1);
02134 NC_GET_DOUBLE("lon", atm->lon, 1);
02135 NC_GET_DOUBLE("lat", atm->lat, 1);
02136
02137 /* Read variables... */
02138 for (int iq = 0; iq < ctl->nq; iq++)
02139     NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
02140
02141 /* Close file... */
02142 NC(nc_close(ncid));
02143
02144 /* Return success... */
02145 return 1;
02146 }

```

**5.21.3.37 read\_clim()** void read\_clim (  
     ctl\_t \* ctl,  
     clim\_t \* clim )

Read climatological data.

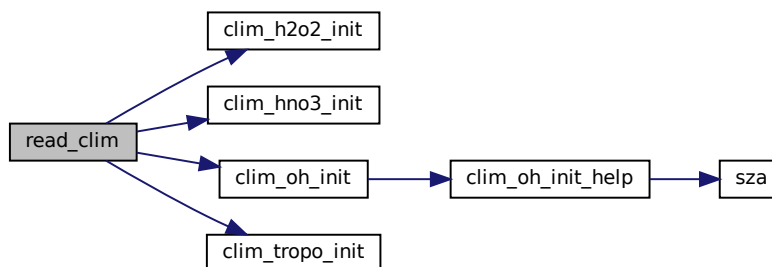
Definition at line 2150 of file libtrac.c.

```

02152 {
02153
02154     /* Set timer... */
02155     SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);
02156
02157     /* Init tropopause climatology... */
02158     clim_tropo_init(clim);
02159
02160     /* Init HNO3 climatology... */
02161     clim_hno3_init(clim);
02162
02163     /* Read OH climatology... */
02164     if (ctl->clim_oh_filename[0] != '-')
02165         clim_oh_init(ctl, clim);
02166
02167     /* Read H2O2 climatology... */
02168     if (ctl->clim_h2o2_filename[0] != '-')
02169         clim_h2o2_init(ctl, clim);
02170 }

```

Here is the call graph for this function:



```

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

```

Read control parameters.

Definition at line 2174 of file [libtrac.c](#).

```

02178     {
02179
02180         /* Set timer... */
02181         SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02182
02183         /* Write info... */
02184         LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02185             "(executable: %s | version: %s | compiled: %s, %s)\n",
02186             argv[0], VERSION, __DATE__, __TIME__);
02187
02188         /* Initialize quantity indices... */
02189         ctl->qnt_idx = -1;
02190         ctl->qnt_ens = -1;
02191         ctl->qnt_stat = -1;
02192         ctl->qnt_m = -1;
02193         ctl->qnt_vmr = -1;
02194         ctl->qnt_rp = -1;
02195         ctl->qnt_rhop = -1;
02196         ctl->qnt_ps = -1;
02197         ctl->qnt_ts = -1;
02198         ctl->qnt_zs = -1;
02199         ctl->qnt_us = -1;
02200         ctl->qnt_vs = -1;
02201         ctl->qnt_pbl = -1;
02202         ctl->qnt_pt = -1;
02203         ctl->qnt_tt = -1;
02204         ctl->qnt_zt = -1;
02205         ctl->qnt_h2ot = -1;
02206         ctl->qnt_z = -1;
02207         ctl->qnt_p = -1;
02208         ctl->qnt_t = -1;
02209         ctl->qnt_rho = -1;
02210         ctl->qnt_u = -1;
02211         ctl->qnt_v = -1;
02212         ctl->qnt_w = -1;
02213         ctl->qnt_h2o = -1;
02214         ctl->qnt_o3 = -1;
02215         ctl->qnt_lwc = -1;
02216         ctl->qnt_iwc = -1;
02217         ctl->qnt_pct = -1;
02218         ctl->qnt_pcb = -1;
02219         ctl->qnt_cl = -1;
02220         ctl->qnt_plcl = -1;
02221         ctl->qnt_plfc = -1;
02222         ctl->qnt_pel = -1;
02223         ctl->qnt_cape = -1;
02224         ctl->qnt_cin = -1;
02225         ctl->qnt_hno3 = -1;
02226         ctl->qnt_oh = -1;
02227         ctl->qnt_vmrimpl = -1;
02228         ctl->qnt_mloss_oh = -1;
02229         ctl->qnt_mloss_h2o2 = -1;
02230         ctl->qnt_mloss_wet = -1;
02231         ctl->qnt_mloss_dry = -1;
02232         ctl->qnt_mloss_decay = -1;
02233         ctl->qnt_psat = -1;
02234         ctl->qnt_psice = -1;
02235         ctl->qnt_pw = -1;
02236         ctl->qnt_sh = -1;
02237         ctl->qnt_rh = -1;
02238         ctl->qnt_rhice = -1;
02239         ctl->qnt_theta = -1;
02240         ctl->qnt_zeta = -1;
02241         ctl->qnt_tvirt = -1;
02242         ctl->qnt_lapse = -1;
02243         ctl->qnt_vh = -1;
02244         ctl->qnt_vz = -1;
02245         ctl->qnt_pv = -1;
02246         ctl->qnt_tdew = -1;
02247         ctl->qnt_tice = -1;
02248         ctl->qnt_tsts = -1;
02249         ctl->qnt_tnat = -1;
02250
02251         /* Read quantities... */

```



```

02252     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02253     if (ctl->nq > NQ)
02254         ERRMSG("Too many quantities!");
02255     for (int iq = 0; iq < ctl->nq; iq++) {
02256
02257         /* Read quantity name and format... */
02258         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02259         scan_ctl(filename, argc, argv, "QNT_LONGNAME", iq, ctl->qnt_name[iq],
02260             ctl->qnt_longname[iq]);
02261         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02262             ctl->qnt_format[iq]);
02263
02264         /* Try to identify quantity... */
02265         SET_QNT(qnt_idx, "idx", "particle index", "-")
02266         SET_QNT(qnt_ens, "ens", "ensemble index", "-")
02267         SET_QNT(qnt_stat, "stat", "station flag", "-")
02268         SET_QNT(qnt_m, "m", "mass", "kg")
02269         SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
02270         SET_QNT(qnt_rp, "rp", "particle radius", "microns")
02271         SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
02272         SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
02273         SET_QNT(qnt_ts, "ts", "surface temperature", "K")
02274         SET_QNT(qnt_zs, "zs", "surface height", "km")
02275         SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
02276         SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
02277         SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
02278         SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
02279         SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
02280         SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
02281         SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
02282         SET_QNT(qnt_z, "z", "geopotential height", "km")
02283         SET_QNT(qnt_p, "p", "pressure", "hPa")
02284         SET_QNT(qnt_t, "t", "temperature", "K")
02285         SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
02286         SET_QNT(qnt_u, "u", "zonal wind", "m/s")
02287         SET_QNT(qnt_v, "v", "meridional wind", "m/s")
02288         SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
02289         SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
02290         SET_QNT(qnt_o3, "o3", "ozone", "ppv")
02291         SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
02292         SET_QNT(qnt_lwc, "lwc", "cloud liquid water content", "kg/kg")
02293         SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
02294         SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
02295         SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
02296         SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
02297         SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
02298         SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
02299         SET_QNT(qnt_cape, "cape", "convective available potential energy",
02300             "J/kg")
02301         SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
02302         SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
02303         SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
02304         SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
02305         SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
02306         SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
02307             "kg")
02308         SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
02309             "kg")
02310         SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
02311             "kg")
02312         SET_QNT(qnt_mloss_decay, "mloss_decay",
02313             "mass loss due to exponential decay", "kg")
02314         SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
02315         SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
02316         SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
02317         SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
02318         SET_QNT(qnt_rh, "rh", "relative humidity", "%")
02319         SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%")
02320         SET_QNT(qnt_theta, "theta", "potential temperature", "K")
02321         SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
02322         SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
02323         SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
02324         SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
02325         SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
02326         SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
02327         SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
02328         SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
02329         SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
02330         SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
02331         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02332     }
02333
02334     /* netCDF I/O parameters... */
02335     ctl->chunkszhint =
02336         (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02337             NULL);
02338     ctl->read_mode =

```

```

02339     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02340
02341     /* Vertical coordinates and velocities... */
02342     ctl->vert_coord_ap =
02343         (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
02344     ctl->vert_coord_met =
02345         (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
02346     ctl->vert_vel =
02347         (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
02348     ctl->clams_met_data =
02349         (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);
02350
02351     /* Time steps of simulation... */
02352     ctl->direction =
02353         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02354     if (ctl->direction != -1 && ctl->direction != 1)
02355         ERRMSG("Set DIRECTION to -1 or 1!");
02356     ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02357     ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02358
02359     /* Meteo data... */
02360     scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02361     ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02362     ctl->met_type =
02363         (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
02364     ctl->met_nc_scale =
02365         (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", NULL);
02366     ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02367     ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02368     ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02369     if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02370         ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02371     ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02372     ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02373     ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02374     if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02375         ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02376     ctl->met_detrend =
02377         scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02378     ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02379     if (ctl->met_np > EP)
02380         ERRMSG("Too many levels!");
02381     for (int ip = 0; ip < ctl->met_np; ip++)
02382         ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02383     ctl->met_geopot_sx =
02384         (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02385     ctl->met_geopot_sy =
02386         (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02387     ctl->met_tropo =
02388         (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02389     if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02390         ERRMSG("Set MET_TROPO = 0 ... 5!");
02391     ctl->met_tropo_lapse =
02392         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02393     ctl->met_tropo_nlev =
02394         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02395     ctl->met_tropo_lapse_sep =
02396         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02397     ctl->met_tropo_nlev_sep =
02398         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02399             NULL);
02400     ctl->met_tropo_pv =
02401         scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02402     ctl->met_tropo_theta =
02403         scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02404     ctl->met_tropo_spline =
02405         (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02406     ctl->met_cloud =
02407         (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02408     if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02409         ERRMSG("Set MET_CLOUD = 0 ... 3!");
02410     ctl->met_cloud_min =
02411         scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
02412     ctl->met_dt_out =
02413         scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02414     ctl->met_cache =
02415         (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02416
02417     /* Sorting... */
02418     ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);
02419
02420     /* Isosurface parameters... */
02421     ctl->isosurf =
02422         (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02423     scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02424
02425     /* Advection parameters... */

```

```

02426     ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
02427     if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
02428         ERRMSG("Set ADVECT to 1, 2, or 4!");
02429     ctl->reflect =
02430         (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02431
02432     /* Diffusion parameters... */
02433     ctl->turb_dx_trop =
02434         scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02435     ctl->turb_dx_strat =
02436         scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02437     ctl->turb_dz_trop =
02438         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02439     ctl->turb_dz_strat =
02440         scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02441     ctl->turb_mesox =
02442         scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02443     ctl->turb_mesoz =
02444         scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02445
02446     /* Convection... */
02447     ctl->conv_cape =
02448         scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02449     ctl->conv_cin =
02450         scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02451     ctl->conv_wmax =
02452         scan_ctl(filename, argc, argv, "CONV_WMAX", -1, "-999", NULL);
02453     ctl->conv_wcape =
02454         (int) scan_ctl(filename, argc, argv, "CONV_WCAPE", -1, "0", NULL);
02455     ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02456     ctl->conv_mix =
02457         (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "0", NULL);
02458     ctl->conv_mix_bot =
02459         (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02460     ctl->conv_mix_top =
02461         (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02462
02463     /* Boundary conditions... */
02464     ctl->bound_mass =
02465         scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02466     ctl->bound_mass_trend =
02467         scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
02468     ctl->bound_vmr =
02469         scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02470     ctl->bound_vmr_trend =
02471         scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
02472     ctl->bound_lat0 =
02473         scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02474     ctl->bound_lat1 =
02475         scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02476     ctl->bound_p0 =
02477         scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02478     ctl->bound_p1 =
02479         scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02480     ctl->bound_dps =
02481         scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02482
02483     /* Species parameters... */
02484     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02485     if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02486         ctl->molmass = 120.907;
02487         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
02488         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
02489     } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02490         ctl->molmass = 137.359;
02491         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
02492         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
02493     } else if (strcasecmp(ctl->species, "CH4") == 0) {
02494         ctl->molmass = 16.043;
02495         ctl->oh_chem_reaction = 2;
02496         ctl->oh_chem[0] = 2.45e-12;
02497         ctl->oh_chem[1] = 1775;
02498         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
02499         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02500     } else if (strcasecmp(ctl->species, "CO") == 0) {
02501         ctl->molmass = 28.01;
02502         ctl->oh_chem_reaction = 3;
02503         ctl->oh_chem[0] = 6.9e-33;
02504         ctl->oh_chem[1] = 2.1;
02505         ctl->oh_chem[2] = 1.1e-12;
02506         ctl->oh_chem[3] = -1.3;
02507         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
02508         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
02509     } else if (strcasecmp(ctl->species, "CO2") == 0) {
02510         ctl->molmass = 44.009;
02511         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
02512         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;

```

```

02513 } else if (strcasecmp(ctl->species, "H2O") == 0) {
02514     ctl->molmass = 18.01528;
02515 } else if (strcasecmp(ctl->species, "N2O") == 0) {
02516     ctl->molmass = 44.013;
02517     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
02518     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
02519 } else if (strcasecmp(ctl->species, "NH3") == 0) {
02520     ctl->molmass = 17.031;
02521     ctl->oh_chem_reaction = 2;
02522     ctl->oh_chem[0] = 1.7e-12;
02523     ctl->oh_chem[1] = 710;
02524     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
02525     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
02526 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02527     ctl->molmass = 63.012;
02528     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
02529     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
02530 } else if (strcasecmp(ctl->species, "NO") == 0) {
02531     ctl->molmass = 30.006;
02532     ctl->oh_chem_reaction = 3;
02533     ctl->oh_chem[0] = 7.1e-31;
02534     ctl->oh_chem[1] = 2.6;
02535     ctl->oh_chem[2] = 3.6e-11;
02536     ctl->oh_chem[3] = 0.1;
02537     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
02538     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02539 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02540     ctl->molmass = 46.005;
02541     ctl->oh_chem_reaction = 3;
02542     ctl->oh_chem[0] = 1.8e-30;
02543     ctl->oh_chem[1] = 3.0;
02544     ctl->oh_chem[2] = 2.8e-11;
02545     ctl->oh_chem[3] = 0.0;
02546     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
02547     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02548 } else if (strcasecmp(ctl->species, "O3") == 0) {
02549     ctl->molmass = 47.997;
02550     ctl->oh_chem_reaction = 2;
02551     ctl->oh_chem[0] = 1.7e-12;
02552     ctl->oh_chem[1] = 940;
02553     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
02554     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
02555 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02556     ctl->molmass = 146.048;
02557     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
02558     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
02559 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02560     ctl->molmass = 64.066;
02561     ctl->oh_chem_reaction = 3;
02562     ctl->oh_chem[0] = 2.9e-31;
02563     ctl->oh_chem[1] = 4.1;
02564     ctl->oh_chem[2] = 1.7e-12;
02565     ctl->oh_chem[3] = -0.2;
02566     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
02567     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
02568 } else {
02569     ctl->molmass =
02570         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02571     ctl->oh_chem_reaction =
02572         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02573     ctl->h2o2_chem_reaction =
02574         (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
02575             NULL);
02576     for (int ip = 0; ip < 4; ip++)
02577         ctl->oh_chem[ip] =
02578             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02579     for (int ip = 0; ip < 1; ip++)
02580         ctl->dry_depo[ip] =
02581             scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02582     ctl->wet_depo_ic_a =
02583         scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
02584     ctl->wet_depo_ic_b =
02585         scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
02586     ctl->wet_depo_bc_a =
02587         scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
02588     ctl->wet_depo_bc_b =
02589         scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
02590     for (int ip = 0; ip < 3; ip++)
02591         ctl->wet_depo_ic_h[ip] =
02592             scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
02593     for (int ip = 0; ip < 1; ip++)
02594         ctl->wet_depo_bc_h[ip] =
02595             scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
02596 }
02597
02598 /* Wet deposition... */
02599 ctl->wet_depo_pre[0] =

```

```

02600     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
02601     ctl->wet_depo_pre[1] =
02602     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
02603     ctl->wet_depo_ic_ret_ratio =
02604     scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
02605     ctl->wet_depo_bc_ret_ratio =
02606     scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);
02607
02608     /* OH chemistry... */
02609     ctl->oh_chem_beta =
02610     scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
02611     scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
02612     "../data/clams_radical_species.nc", ctl->clim_oh_filename);
02613
02614     /* H2O2 chemistry... */
02615     ctl->h2o2_chem_cc =
02616     scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
02617     scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
02618     "../data/cams_H2O2.nc", ctl->clim_h2o2_filename);
02619
02620     /* Exponential decay... */
02621     ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02622     ctl->tdec_strat =
02623     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02624
02625     /* PSC analysis... */
02626     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02627     ctl->psc_hno3 =
02628     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02629
02630     /* Output of atmospheric data... */
02631     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02632     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02633     ctl->atm_dt_out =
02634     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02635     ctl->atm_filter =
02636     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02637     ctl->atm_stride =
02638     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02639     ctl->atm_type =
02640     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02641
02642     /* Output of CSI data... */
02643     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02644     ctl->csi_dt_out =
02645     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02646     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02647     ctl->csi_obsmin =
02648     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02649     ctl->csi_modmin =
02650     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02651     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02652     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02653     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02654     ctl->csi_lon0 =
02655     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02656     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02657     ctl->csi_nx =
02658     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02659     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02660     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02661     ctl->csi_ny =
02662     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02663
02664     /* Output of ensemble data... */
02665     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02666     ctl->ens_dt_out =
02667     scan_ctl(filename, argc, argv, "ENS_DT_OUT", -1, "86400", NULL);
02668
02669     /* Output of grid data... */
02670     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02671     ctl->grid_basename);
02672     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02673     ctl->grid_dt_out =
02674     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02675     ctl->grid_sparse =
02676     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02677     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02678     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02679     ctl->grid_nz =
02680     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02681     ctl->grid_lon0 =
02682     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02683     ctl->grid_lon1 =
02684     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02685     ctl->grid_nx =
02686     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);

```

```

02687   ctl->grid_lat0 =
02688       scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02689   ctl->grid_lat1 =
02690       scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02691   ctl->grid_ny =
02692       (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02693   ctl->grid_type =
02694       (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", NULL);
02695
02696   /* Output of profile data... */
02697   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02698           ctl->prof_basename);
02699   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02700   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02701   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02702   ctl->prof_nz =
02703       (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02704   ctl->prof_lon0 =
02705       scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02706   ctl->prof_lon1 =
02707       scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02708   ctl->prof_nx =
02709       (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02710   ctl->prof_lat0 =
02711       scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02712   ctl->prof_lat1 =
02713       scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02714   ctl->prof_ny =
02715       (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02716
02717   /* Output of sample data... */
02718   scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02719           ctl->sample_basename);
02720   scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02721           ctl->sample_obsfile);
02722   ctl->sample_dx =
02723       scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02724   ctl->sample_dz =
02725       scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02726
02727   /* Output of station data... */
02728   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02729           ctl->stat_basename);
02730   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02731   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02732   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02733   ctl->stat_t0 =
02734       scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02735   ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02736 }

```

Here is the call graph for this function:



**5.21.3.39 read\_met()** int read\_met (

```

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

```

Read meteo data file.

Definition at line 2740 of file libtrac.c.

```

02744     {
02745
02746     /* Write info... */
02747     LOG(1, "Read meteo data: %s", filename);
02748
02749     /* Read netCDF data... */
02750     if (ctl->met_type == 0) {
02751
02752         int ncid;
02753
02754         /* Open netCDF file... */
02755         if (nc_open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02756             NC_NOERR) {
02757             WARN("Cannot open file!");
02758             return 0;
02759         }
02760
02761         /* Read coordinates of meteo data... */
02762         read_met_grid(filename, ncid, ctl, met);
02763
02764         /* Read meteo data on vertical levels... */
02765         read_met_levels(ncid, ctl, met);
02766
02767         /* Extrapolate data for lower boundary... */
02768         read_met_extrapolate(met);
02769
02770         /* Read surface data... */
02771         read_met_surface(ncid, met, ctl);
02772
02773         /* Create periodic boundary conditions... */
02774         read_met_periodic(met);
02775
02776         /* Downsampling... */
02777         read_met_sample(ctl, met);
02778
02779         /* Calculate geopotential heights... */
02780         read_met_geopot(ctl, met);
02781
02782         /* Calculate potential vorticity... */
02783         read_met_pv(met);
02784
02785         /* Calculate boundary layer data... */
02786         read_met_pbl(met);
02787
02788         /* Calculate tropopause data... */
02789         read_met_tropo(ctl, clim, met);
02790
02791         /* Calculate cloud properties... */
02792         read_met_cloud(ctl, met);
02793
02794         /* Calculate convective available potential energy... */
02795         read_met_cape(clim, met);
02796
02797         /* Detrending... */
02798         read_met_detrend(ctl, met);
02799
02800         /* Close file... */
02801         NC(nc_close(ncid));
02802     }
02803
02804     /* Read binary data... */
02805     else if (ctl->met_type >= 1 && ctl->met_type <= 4) {
02806
02807         FILE *in;
02808
02809         double r;
02810
02811         int year, mon, day, hour, min, sec;
02812
02813         /* Set timer... */
02814         SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);
02815
02816         /* Open file... */
02817         if (!(in = fopen(filename, "r"))) {
02818             WARN("Cannot open file!");
02819             return 0;
02820         }
02821
02822         /* Check type of binary data... */
02823         int met_type;
02824         FREAD(&met_type, int,
02825             1,
02826             in);
02827         if (met_type != ctl->met_type)
02828             ERRMSG("Wrong MET_TYPE of binary data!");
02829

```

```

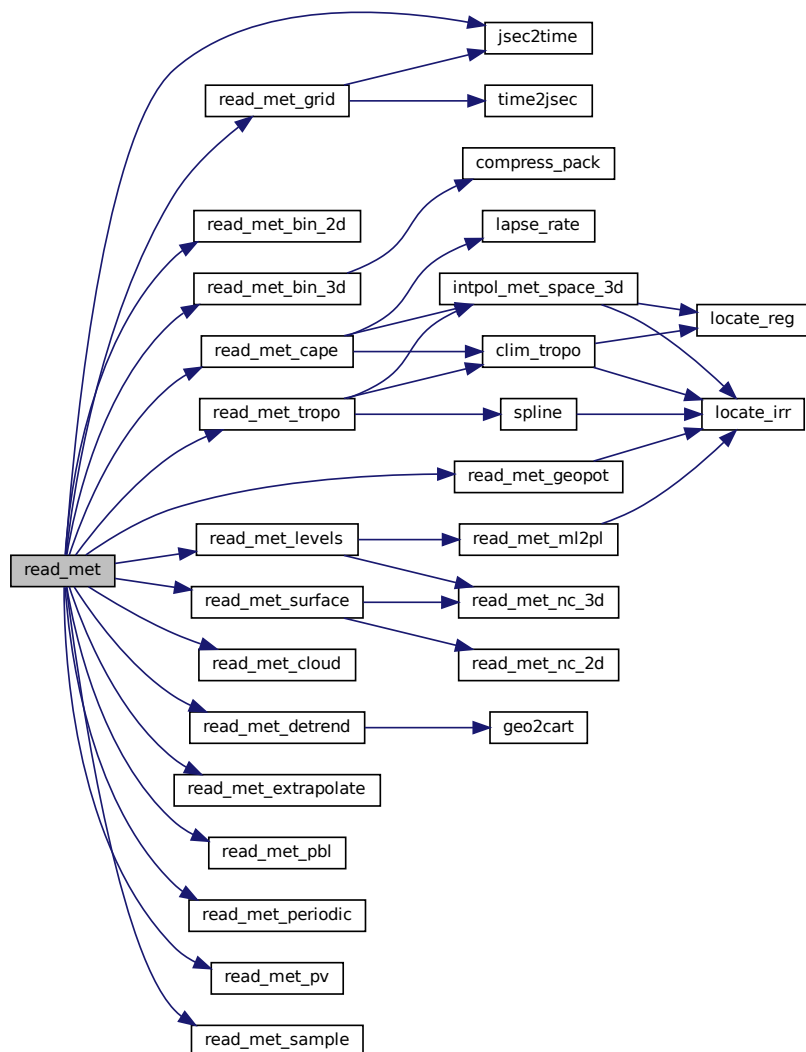
02830      /* Check version of binary data... */
02831      int version;
02832      FREAD(&version, int,
02833           1,
02834           in);
02835      if (version != 100)
02836          ERRMSG("Wrong version of binary data!");
02837
02838      /* Read time... */
02839      FREAD(&met->time, double,
02840           1,
02841           in);
02842      jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
02843      LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
02844          met->time, year, mon, day, hour, min);
02845      if (year < 1900 || year > 2100 || mon < 1 || mon > 12
02846          || day < 1 || day > 31 || hour < 0 || hour > 23)
02847          ERRMSG("Error while reading time!");
02848
02849      /* Read dimensions... */
02850      FREAD(&met->nx, int,
02851           1,
02852           in);
02853      LOG(2, "Number of longitudes: %d", met->nx);
02854      if (met->nx < 2 || met->nx > EX)
02855          ERRMSG("Number of longitudes out of range!");
02856
02857      FREAD(&met->ny, int,
02858           1,
02859           in);
02860      LOG(2, "Number of latitudes: %d", met->ny);
02861      if (met->ny < 2 || met->ny > EY)
02862          ERRMSG("Number of latitudes out of range!");
02863
02864      FREAD(&met->np, int,
02865           1,
02866           in);
02867      LOG(2, "Number of levels: %d", met->np);
02868      if (met->np < 2 || met->np > EP)
02869          ERRMSG("Number of levels out of range!");
02870
02871      /* Read grid... */
02872      FREAD(met->lon, double,
02873           (size_t) met->nx,
02874           in);
02875      LOG(2, "Longitudes: %g, %g ... %g deg",
02876          met->lon[0], met->lon[1], met->lon[met->nx - 1]);
02877
02878      FREAD(met->lat, double,
02879           (size_t) met->ny,
02880           in);
02881      LOG(2, "Latitudes: %g, %g ... %g deg",
02882          met->lat[0], met->lat[1], met->lat[met->ny - 1]);
02883
02884      FREAD(met->p, double,
02885           (size_t) met->np,
02886           in);
02887      LOG(2, "Altitude levels: %g, %g ... %g km",
02888          Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
02889      LOG(2, "Pressure levels: %g, %g ... %g hPa",
02890          met->p[0], met->p[1], met->p[met->np - 1]);
02891
02892      /* Read surface data... */
02893      read_met_bin_2d(in, met, met->ps, "PS");
02894      read_met_bin_2d(in, met, met->ts, "TS");
02895      read_met_bin_2d(in, met, met->zs, "ZS");
02896      read_met_bin_2d(in, met, met->us, "US");
02897      read_met_bin_2d(in, met, met->vs, "VS");
02898      read_met_bin_2d(in, met, met->pbl, "PBL");
02899      read_met_bin_2d(in, met, met->pt, "PT");
02900      read_met_bin_2d(in, met, met->tt, "TT");
02901      read_met_bin_2d(in, met, met->zt, "ZT");
02902      read_met_bin_2d(in, met, met->h2ot, "H2OT");
02903      read_met_bin_2d(in, met, met->pct, "PCT");
02904      read_met_bin_2d(in, met, met->pcb, "PCB");
02905      read_met_bin_2d(in, met, met->cl, "CL");
02906      read_met_bin_2d(in, met, met->plcl, "PLCL");
02907      read_met_bin_2d(in, met, met->plfc, "PLFC");
02908      read_met_bin_2d(in, met, met->pel, "PEL");
02909      read_met_bin_2d(in, met, met->cape, "CAPE");
02910      read_met_bin_2d(in, met, met->cin, "CIN");
02911
02912      /* Read level data... */
02913      read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
02914      read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
02915      read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
02916      read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);

```



```
02917     read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
02918     read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
02919     read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
02920     read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
02921     read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
02922     read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);
02923
02924     /* Read final flag... */
02925     int final;
02926     FREAD(&final, int,
02927          1,
02928          in);
02929     if (final != 999)
02930         ERRMSG("Error while reading binary data!");
02931
02932     /* Close file... */
02933     fclose(in);
02934 }
02935
02936 /* Not implemented... */
02937 else
02938     ERRMSG("MET_TYPE not implemented!");
02939
02940 /* Copy wind data to cache... */
02941 #ifdef UVW
02942 #pragma omp parallel for default(shared) collapse(2)
02943     for (int ix = 0; ix < met->nx; ix++)
02944         for (int iy = 0; iy < met->ny; iy++)
02945             for (int ip = 0; ip < met->np; ip++) {
02946                 met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
02947                 met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
02948                 met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
02949             }
02950 #endif
02951
02952 /* Return success... */
02953 return 1;
02954 }
```

Here is the call graph for this function:



**5.21.3.40 read\_met\_bin\_2d()** void read\_met\_bin\_2d (  
 FILE \* out,  
 met\_t \* met,  
 float var[EX][EY],  
 char \* varname )

Read 2-D meteo variable.

Definition at line 2958 of file libtrac.c.

```

02962     {
02963
02964     float *help;
02965
02966     /* Allocate... */
02967     ALLOC(help, float,
02968           EX * EY);
  
```

```

02969
02970 /* Read uncompressed... */
02971 LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
02972 FREAD(help, float,
02973       (size_t) (met->nx * met->ny),
02974       in);
02975
02976 /* Copy data... */
02977 for (int ix = 0; ix < met->nx; ix++)
02978     for (int iy = 0; iy < met->ny; iy++)
02979         var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];
02980
02981 /* Free... */
02982 free(help);
02983 }

```

#### 5.21.3.41 read\_met\_bin\_3d() void read\_met\_bin\_3d (

```

FILE * in,
ctl_t * ctl,
met_t * met,
float var[EX][EY][EP],
char * varname,
int precision,
double tolerance )

```

Read 3-D meteo variable.

Definition at line 2987 of file libtrac.c.

```

02994 {
02995
02996     float *help;
02997
02998     /* Allocate... */
02999     ALLOC(help, float,
03000          EX * EY * EP);
03001
03002     /* Read uncompressed data... */
03003     if (ctl->met_type == 1) {
03004         LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
03005         FREAD(help, float,
03006              (size_t) (met->nx * met->ny * met->np),
03007              in);
03008     }
03009
03010     /* Read packed data... */
03011     else if (ctl->met_type == 2)
03012         compress_pack(varname, help, (size_t) (met->ny * met->nx),
03013                      (size_t) met->np, 1, in);
03014
03015     /* Read zfp data... */
03016     else if (ctl->met_type == 3) {
03017 #ifdef ZFP
03018         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
03019                     tolerance, 1, in);
03020 #else
03021         ERRMSG("zfp compression not supported!");
03022         LOG(3, "%d %g", precision, tolerance);
03023 #endif
03024     }
03025
03026     /* Read zstd data... */
03027     else if (ctl->met_type == 4) {
03028 #ifdef ZSTD
03029         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
03030                      in);
03031 #else
03032         ERRMSG("zstd compression not supported!");
03033 #endif
03034     }
03035
03036     /* Copy data... */
03037 #pragma omp parallel for default(shared) collapse(2)
03038     for (int ix = 0; ix < met->nx; ix++)
03039         for (int iy = 0; iy < met->ny; iy++)
03040             for (int ip = 0; ip < met->np; ip++)
03041                 var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];

```

```

03042
03043  /* Free... */
03044  free(help);
03045 }

```

Here is the call graph for this function:



**5.21.3.42 read\_met\_cape()** void read\_met\_cape (  
     clim\_t \* clim,  
     met\_t \* met )

Calculate convective available potential energy.

Definition at line 3049 of file libtrac.c.

```

03051     {
03052
03053     /* Set timer... */
03054     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
03055     LOG(2, "Calculate CAPE...");
03056
03057     /* Vertical spacing (about 100 m)... */
03058     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
03059
03060     /* Loop over columns... */
03061     #pragma omp parallel for default(shared) collapse(2)
03062     for (int ix = 0; ix < met->nx; ix++)
03063         for (int iy = 0; iy < met->ny; iy++) {
03064
03065             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
03066             int n = 0;
03067             double h2o = 0, t, theta = 0;
03068             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
03069             double ptop = pbot - 50.;
03070             for (int ip = 0; ip < met->np; ip++) {
03071                 if (met->p[ip] <= pbot) {
03072                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
03073                     h2o += met->h2o[ix][iy][ip];
03074                     n++;
03075                 }
03076                 if (met->p[ip] < ptop && n > 0)
03077                     break;
03078             }
03079             theta /= n;
03080             h2o /= n;
03081
03082             /* Cannot compute anything if water vapor is missing... */
03083             met->plcl[ix][iy] = GSL_NAN;
03084             met->plfc[ix][iy] = GSL_NAN;
03085             met->pel[ix][iy] = GSL_NAN;
03086             met->cape[ix][iy] = GSL_NAN;
03087             met->cin[ix][iy] = GSL_NAN;
03088             if (h2o <= 0)
03089                 continue;
03090
03091             /* Find lifted condensation level (LCL)... */
03092             ptop = P(20.);
03093             pbot = met->ps[ix][iy];
03094             do {
03095                 met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
03096                 t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
03097                 if (RH(met->plcl[ix][iy], t, h2o) > 100.)

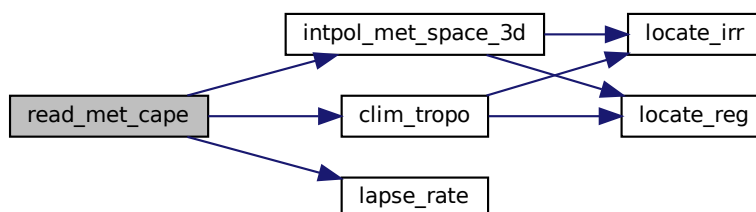
```

```

03098     ptop = met->plcl[ix][iy];
03099     else
03100     pbot = met->plcl[ix][iy];
03101 } while (pbot - ptop > 0.1);
03102
03103 /* Calculate CIN up to LCL... */
03104 INTPOL_INIT;
03105 double dcape, dz, h2o_env, t_env;
03106 double p = met->ps[ix][iy];
03107 met->cape[ix][iy] = met->cin[ix][iy] = 0;
03108 do {
03109     dz = dz0 * TVIRT(t, h2o);
03110     p /= pfac;
03111     t = theta / pow(1000. / p, 0.286);
03112     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03113                        &t_env, ci, cw, 1);
03114     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03115                        &h2o_env, ci, cw, 0);
03116     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03117             TVIRT(t_env, h2o_env) * dz;
03118     if (dcape < 0)
03119         met->cin[ix][iy] += fabsf((float) dcape);
03120 } while (p > met->plcl[ix][iy]);
03121
03122 /* Calculate level of free convection (LFC), equilibrium level (EL),
03123    and convective available potential energy (CAPE)... */
03124 dcape = 0;
03125 p = met->plcl[ix][iy];
03126 t = theta / pow(1000. / p, 0.286);
03127 ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
03128 do {
03129     dz = dz0 * TVIRT(t, h2o);
03130     p /= pfac;
03131     t -= lapse_rate(t, h2o) * dz;
03132     double psat = PSAT(t);
03133     h2o = psat / (p - (1. - EPS) * psat);
03134     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03135                        &t_env, ci, cw, 1);
03136     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03137                        &h2o_env, ci, cw, 0);
03138     double dcape_old = dcape;
03139     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03140             TVIRT(t_env, h2o_env) * dz;
03141     if (dcape > 0) {
03142         met->cape[ix][iy] += (float) dcape;
03143         if (!isfinite(met->plfc[ix][iy]))
03144             met->plfc[ix][iy] = (float) p;
03145     } else if (dcape_old > 0)
03146         met->pel[ix][iy] = (float) p;
03147     if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
03148         met->cin[ix][iy] += fabsf((float) dcape);
03149 } while (p > ptop);
03150
03151 /* Check results... */
03152 if (!isfinite(met->plfc[ix][iy]))
03153     met->cin[ix][iy] = GSL_NAN;
03154 }
03155 }

```

Here is the call graph for this function:



**5.21.3.43 read\_met\_cloud()** void read\_met\_cloud (

```

    ctl_t * ctl,
    met_t * met )

```

Calculate cloud properties.

Definition at line 3159 of file libtrac.c.

```

03161     {
03162
03163     /* Set timer... */
03164     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
03165     LOG(2, "Calculate cloud data...");
03166
03167     /* Loop over columns... */
03168     #pragma omp parallel for default(shared) collapse(2)
03169     for (int ix = 0; ix < met->nx; ix++)
03170         for (int iy = 0; iy < met->ny; iy++) {
03171
03172         /* Init... */
03173         met->pct[ix][iy] = GSL_NAN;
03174         met->pcb[ix][iy] = GSL_NAN;
03175         met->cl[ix][iy] = 0;
03176
03177         /* Loop over pressure levels... */
03178         for (int ip = 0; ip < met->np - 1; ip++) {
03179
03180             /* Check pressure... */
03181             if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
03182                 continue;
03183
03184             /* Check ice water and liquid water content... */
03185             if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
03186                 || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {
03187
03188                 /* Get cloud top pressure ... */
03189                 met->pct[ix][iy]
03190                     = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
03191
03192                 /* Get cloud bottom pressure ... */
03193                 if (!isfinite(met->pcb[ix][iy]))
03194                     met->pcb[ix][iy]
03195                         = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
03196             }
03197
03198             /* Get cloud water... */
03199             met->cl[ix][iy] += (float)
03200                 (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
03201                     + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
03202                 * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
03203         }
03204     }
03205 }

```

**5.21.3.44 read\_met\_detrend()** void read\_met\_detrend (

```

    ctl_t * ctl,
    met_t * met )

```

Apply detrending method to temperature and winds.

Definition at line 3209 of file libtrac.c.

```

03211     {
03212
03213     met_t *help;
03214
03215     /* Check parameters... */
03216     if (ctl->met_detrend <= 0)
03217         return;
03218
03219     /* Set timer... */
03220     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
03221     LOG(2, "Detrend meteo data...");
03222
03223     /* Allocate... */
03224     ALLOC(help, met_t, 1);
03225
03226     /* Calculate standard deviation... */

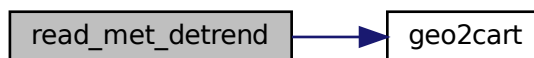
```

```

03227 double sigma = ctl->met_detrend / 2.355;
03228 double tssq = 2. * SQR(sigma);
03229
03230 /* Calculate box size in latitude... */
03231 int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03232 sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03233
03234 /* Calculate background... */
03235 #pragma omp parallel for default(shared) collapse(2)
03236 for (int ix = 0; ix < met->nx; ix++) {
03237     for (int iy = 0; iy < met->ny; iy++) {
03238
03239         /* Calculate Cartesian coordinates... */
03240         double x0[3];
03241         geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03242
03243         /* Calculate box size in longitude... */
03244         int sx =
03245             (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03246                 fabs(met->lon[1] - met->lon[0]));
03247         sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03248
03249         /* Init... */
03250         float wsum = 0;
03251         for (int ip = 0; ip < met->np; ip++) {
03252             help->t[ix][iy][ip] = 0;
03253             help->u[ix][iy][ip] = 0;
03254             help->v[ix][iy][ip] = 0;
03255             help->w[ix][iy][ip] = 0;
03256         }
03257
03258         /* Loop over neighboring grid points... */
03259         for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03260             int ix3 = ix2;
03261             if (ix3 < 0)
03262                 ix3 += met->nx;
03263             else if (ix3 >= met->nx)
03264                 ix3 -= met->nx;
03265             for (int iy2 = GSL_MAX(iy - sy, 0);
03266                 iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03267
03268                 /* Calculate Cartesian coordinates... */
03269                 double x1[3];
03270                 geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03271
03272                 /* Calculate weighting factor... */
03273                 float w = (float) exp(-DIST2(x0, x1) / tssq);
03274
03275                 /* Add data... */
03276                 wsum += w;
03277                 for (int ip = 0; ip < met->np; ip++) {
03278                     help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03279                     help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03280                     help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03281                     help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03282                 }
03283             }
03284         }
03285
03286         /* Normalize... */
03287         for (int ip = 0; ip < met->np; ip++) {
03288             help->t[ix][iy][ip] /= wsum;
03289             help->u[ix][iy][ip] /= wsum;
03290             help->v[ix][iy][ip] /= wsum;
03291             help->w[ix][iy][ip] /= wsum;
03292         }
03293     }
03294 }
03295
03296 /* Subtract background... */
03297 #pragma omp parallel for default(shared) collapse(3)
03298 for (int ix = 0; ix < met->nx; ix++)
03299     for (int iy = 0; iy < met->ny; iy++)
03300         for (int ip = 0; ip < met->np; ip++) {
03301             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03302             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03303             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03304             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03305         }
03306
03307 /* Free... */
03308 free(help);
03309 }

```

Here is the call graph for this function:



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

Extrapolate meteo data at lower boundary.

Definition at line 3313 of file libtrac.c.

```

03314     {
03315
03316     /* Set timer... */
03317     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03318     LOG(2, "Extrapolate meteo data...");
03319
03320     /* Loop over columns... */
03321     #pragma omp parallel for default(shared) collapse(2)
03322     for (int ix = 0; ix < met->nx; ix++)
03323     for (int iy = 0; iy < met->ny; iy++) {
03324
03325         /* Find lowest valid data point... */
03326         int ip0;
03327         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03328             if (!isfinite(met->t[ix][iy][ip0])
03329                 || !isfinite(met->u[ix][iy][ip0])
03330                 || !isfinite(met->v[ix][iy][ip0])
03331                 || !isfinite(met->w[ix][iy][ip0]))
03332                 break;
03333
03334         /* Extrapolate... */
03335         for (int ip = ip0; ip >= 0; ip--) {
03336             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03337             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03338             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03339             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03340             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03341             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03342             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03343             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03344         }
03345     }
03346 }
  
```

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

Calculate geopotential heights.

Definition at line 3350 of file libtrac.c.

```

03352     {
03353
03354     static float help[EP][EX][EY];
03355
  
```



```

03356 double logp[EP];
03357
03358 int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03359
03360 /* Set timer... */
03361 SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03362 LOG(2, "Calculate geopotential heights...");
03363
03364 /* Calculate log pressure... */
03365 #pragma omp parallel for default(shared)
03366 for (int ip = 0; ip < met->np; ip++)
03367     logp[ip] = log(met->p[ip]);
03368
03369 /* Apply hydrostatic equation to calculate geopotential heights... */
03370 #pragma omp parallel for default(shared) collapse(2)
03371 for (int ix = 0; ix < met->nx; ix++)
03372     for (int iy = 0; iy < met->ny; iy++) {
03373
03374         /* Get surface height and pressure... */
03375         double zs = met->zs[ix][iy];
03376         double lnps = log(met->ps[ix][iy]);
03377
03378         /* Get temperature and water vapor vmr at the surface... */
03379         int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03380         double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03381             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03382         double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03383             met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03384
03385         /* Upper part of profile... */
03386         met->z[ix][iy][ip0 + 1]
03387             = (float) (zs +
03388                 ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03389                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03390         for (int ip = ip0 + 2; ip < met->np; ip++)
03391             met->z[ix][iy][ip]
03392                 = (float) (met->z[ix][iy][ip - 1] +
03393                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03394                         met->h2o[ix][iy][ip - 1], logp[ip],
03395                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03396
03397         /* Lower part of profile... */
03398         met->z[ix][iy][ip0]
03399             = (float) (zs +
03400                 ZDIFF(lnps, ts, h2os, logp[ip0],
03401                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03402         for (int ip = ip0 - 1; ip >= 0; ip--)
03403             met->z[ix][iy][ip]
03404                 = (float) (met->z[ix][iy][ip + 1] +
03405                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03406                         met->h2o[ix][iy][ip + 1], logp[ip],
03407                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03408     }
03409
03410 /* Check control parameters... */
03411 if (dx == 0 || dy == 0)
03412     return;
03413
03414 /* Default smoothing parameters... */
03415 if (dx < 0 || dy < 0) {
03416     if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03417         dx = 3;
03418         dy = 2;
03419     } else {
03420         dx = 6;
03421         dy = 4;
03422     }
03423 }
03424
03425 /* Calculate weights for smoothing... */
03426 float ws[dx + 1][dy + 1];
03427 #pragma omp parallel for default(shared) collapse(2)
03428 for (int ix = 0; ix <= dx; ix++)
03429     for (int iy = 0; iy < dy; iy++)
03430         ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03431             * (1.0f - (float) iy / (float) dy);
03432
03433 /* Copy data... */
03434 #pragma omp parallel for default(shared) collapse(3)
03435 for (int ix = 0; ix < met->nx; ix++)
03436     for (int iy = 0; iy < met->ny; iy++)
03437         for (int ip = 0; ip < met->np; ip++)
03438             help[ip][ix][iy] = met->z[ix][iy][ip];
03439
03440 /* Horizontal smoothing... */
03441 #pragma omp parallel for default(shared) collapse(3)
03442 for (int ip = 0; ip < met->np; ip++)

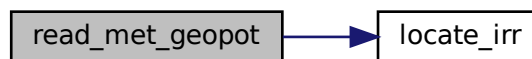
```

```

03443     for (int ix = 0; ix < met->nx; ix++)
03444         for (int iy = 0; iy < met->ny; iy++) {
03445             float res = 0, wsum = 0;
03446             int iy0 = GSL_MAX(iy - dy + 1, 0);
03447             int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03448             for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03449                 int ix3 = ix2;
03450                 if (ix3 < 0)
03451                     ix3 += met->nx;
03452                 else if (ix3 >= met->nx)
03453                     ix3 -= met->nx;
03454                 for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03455                     if (isfinite(help[ip][ix3][iy2])) {
03456                         float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03457                         res += w * help[ip][ix3][iy2];
03458                         wsum += w;
03459                     }
03460             }
03461             if (wsum > 0)
03462                 met->z[ix][iy][ip] = res / wsum;
03463             else
03464                 met->z[ix][iy][ip] = GSL_NAN;
03465         }
03466     }

```

Here is the call graph for this function:



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

```

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

```

Read coordinates of meteo data.

Definition at line 3470 of file libtrac.c.

```

03474     {
03475
03476         char levname[LEN], tstr[10];
03477
03478         double rtime, r2;
03479
03480         int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;
03481
03482         size_t np;
03483
03484         /* Set timer... */
03485         SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03486         LOG(2, "Read meteo grid information...");
03487
03488         /* MPTRAC meteo files... */
03489         if (ctl->clams_met_data == 0) {
03490
03491             /* Get time from filename... */
03492             size_t len = strlen(filename);
03493             sprintf(tstr, "%.4s", &filename[len - 16]);
03494             year = atoi(tstr);
03495             sprintf(tstr, "%.2s", &filename[len - 11]);
03496             mon = atoi(tstr);
03497             sprintf(tstr, "%.2s", &filename[len - 8]);

```

```

03498     day = atoi(tstr);
03499     sprintf(tstr, "%.2s", &filename[len - 5]);
03500     hour = atoi(tstr);
03501     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03502
03503     /* Check time information from data file... */
03504     if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03505         NC(nc_get_var_double(ncid, varid, &rtime));
03506         if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
03507             WARN("Time information in meteo file does not match filename!");
03508     } else
03509         WARN("Time information in meteo file is missing!");
03510 }
03511
03512 /* CLaMS meteo files... */
03513 else {
03514
03515     /* Read time from file... */
03516     NC_GET_DOUBLE("time", &rtime, 0);
03517
03518     /* Get time from filename (considering the century)... */
03519     if (rtime < 0)
03520         sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
03521     else
03522         sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
03523     year = atoi(tstr);
03524     sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
03525     mon = atoi(tstr);
03526     sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
03527     day = atoi(tstr);
03528     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03529     hour = atoi(tstr);
03530     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03531 }
03532
03533 /* Check time... */
03534 if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03535     || day < 1 || day > 31 || hour < 0 || hour > 23)
03536     ERRMSG("Cannot read time from filename!");
03537 jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03538 LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03539     met->time, year2, mon2, day2, hour2, min2);
03540
03541 /* Get grid dimensions... */
03542 NC_INQ_DIM("lon", &met->nx, 2, EX);
03543 LOG(2, "Number of longitudes: %d", met->nx);
03544
03545 NC_INQ_DIM("lat", &met->ny, 2, EY);
03546 LOG(2, "Number of latitudes: %d", met->ny);
03547
03548 if (ctl->vert_coord_meteo == 0) {
03549     sprintf(levname, "lev");
03550 } else {
03551     sprintf(levname, "hybrid");
03552     printf("Meteorological data is in hybrid coordinates.");
03553 }
03554 NC_INQ_DIM(levname, &met->np, 1, EP);
03555 if (met->np == 1) {
03556     int dimid;
03557     sprintf(levname, "lev_2");
03558     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03559         sprintf(levname, "plev");
03560         nc_inq_dimid(ncid, levname, &dimid);
03561     }
03562     NC(nc_inq_dimlen(ncid, dimid, &np));
03563     met->np = (int) np;
03564 }
03565 LOG(2, "Number of levels: %d", met->np);
03566 if (met->np < 2 || met->np > EP)
03567     ERRMSG("Number of levels out of range!");
03568
03569 /* Read longitudes and latitudes... */
03570 NC_GET_DOUBLE("lon", met->lon, 1);
03571 LOG(2, "Longitudes: %g, %g ... %g deg",
03572     met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03573 NC_GET_DOUBLE("lat", met->lat, 1);
03574 LOG(2, "Latitudes: %g, %g ... %g deg",
03575     met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03576
03577 /* Read pressure levels... */
03578 if (ctl->met_np <= 0) {
03579     NC_GET_DOUBLE(levname, met->p, 1);
03580     for (int ip = 0; ip < met->np; ip++)
03581         met->p[ip] /= 100.;
03582     LOG(2, "Altitude levels: %g, %g ... %g km",
03583         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03584     LOG(2, "Pressure levels: %g, %g ... %g hPa",

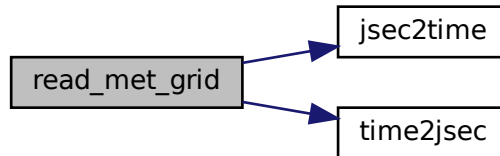
```

```

03585         met->p[0], met->p[1], met->p[met->np - 1]);
03586     }
03587 }

```

Here is the call graph for this function:



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

```

    int ncid,
    ctl_t * ctl,
    met_t * met )

```

Read meteo data on vertical levels.

Definition at line 3591 of file libtrac.c.

```

03594     {
03595
03596         /* Set timer... */
03597         SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03598         LOG(2, "Read level data...");
03599
03600         /* MPTRAC meteo data... */
03601         if (ctl->clams_met_data == 0) {
03602
03603             /* Read meteo data... */
03604             if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
03605                 ERRMSG("Cannot read temperature!");
03606             if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03607                 ERRMSG("Cannot read zonal wind!");
03608             if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03609                 ERRMSG("Cannot read meridional wind!");
03610             if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
03611                 WARN("Cannot read vertical velocity!");
03612             if (!read_met_nc_3d
03613                 (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03614                 WARN("Cannot read specific humidity!");
03615             if (!read_met_nc_3d
03616                 (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03617                 WARN("Cannot read ozone data!");
03618             if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03619                 if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03620                     WARN("Cannot read cloud liquid water content!");
03621                 if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03622                     WARN("Cannot read cloud ice water content!");
03623             }
03624             if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03625                 if (!read_met_nc_3d
03626                     (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03627                     ctl->met_cloud == 2))
03628                     WARN("Cannot read cloud rain water content!");
03629                 if (!read_met_nc_3d
03630                     (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03631                     ctl->met_cloud == 2))
03632                     WARN("Cannot read cloud snow water content!");
03633             }
03634         }

```

```

03635      /* Transfer from model levels to pressure levels... */
03636      if (ctl->met_np > 0) {
03637
03638          /* Read pressure on model levels... */
03639          if (!read_met_nc_3d(ncid, "p1", "PL", ctl, met, met->p1, 0.01f, 1))
03640              ERRMSG("Cannot read pressure on model levels!");
03641
03642          /* Vertical interpolation from model to pressure levels... */
03643          read_met_ml2pl(ctl, met, met->t);
03644          read_met_ml2pl(ctl, met, met->u);
03645          read_met_ml2pl(ctl, met, met->v);
03646          read_met_ml2pl(ctl, met, met->w);
03647          read_met_ml2pl(ctl, met, met->h2o);
03648          read_met_ml2pl(ctl, met, met->o3);
03649          read_met_ml2pl(ctl, met, met->lwc);
03650          read_met_ml2pl(ctl, met, met->iwc);
03651
03652          /* Set new pressure levels... */
03653          met->np = ctl->met_np;
03654          for (int ip = 0; ip < met->np; ip++)
03655              met->p[ip] = ctl->met_p[ip];
03656      }
03657  }
03658
03659  /* CLaMS meteo data... */
03660  else if (ctl->clams_meteo_data == 1) {
03661
03662      /* Read meteorological data... */
03663      if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
03664          ERRMSG("Cannot read temperature!");
03665      if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03666          ERRMSG("Cannot read zonal wind!");
03667      if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03668          ERRMSG("Cannot read meridional wind!");
03669      if (!read_met_nc_3d(ncid, "w", "OMEGA", ctl, met, met->w, 0.01f, 1))
03670          WARN("Cannot read vertical velocity!");
03671      if (!read_met_nc_3d(ncid, "ZETA", "zeta", ctl, met, met->zeta, 1.0, 1))
03672          WARN("Cannot read ZETA in meteo data!");
03673      if (ctl->vert_vel == 1) {
03674          if (!read_met_nc_3d
03675              (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
03676               0.00001157407f, 1)) {
03677              if (!read_met_nc_3d
03678                  (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
03679                   0.00001157407f, 1)) {
03680                  WARN("Cannot read vertical velocity!");
03681              }
03682          }
03683      }
03684
03685      if (!read_met_nc_3d
03686          (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03687          WARN("Cannot read specific humidity!");
03688      if (!read_met_nc_3d
03689          (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03690          WARN("Cannot read ozone data!");
03691      if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03692          if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03693              WARN("Cannot read cloud liquid water content!");
03694          if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03695              WARN("Cannot read cloud ice water content!");
03696      }
03697      if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03698          if (!read_met_nc_3d
03699              (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03700               ctl->met_cloud == 2))
03701              WARN("Cannot read cloud rain water content!");
03702          if (!read_met_nc_3d
03703              (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03704               ctl->met_cloud == 2))
03705              WARN("Cannot read cloud snow water content!");
03706      }
03707
03708      /* Transfer from model levels to pressure levels... */
03709      if (ctl->met_np > 0) {
03710
03711          /* Read pressure on model levels... */
03712          if (!read_met_nc_3d(ncid, "p1", "PRESS", ctl, met, met->p1, 1.0, 1))
03713              ERRMSG("Cannot read pressure on model levels!");
03714
03715          /* Vertical interpolation from model to pressure levels... */
03716          read_met_ml2pl(ctl, met, met->t);
03717          read_met_ml2pl(ctl, met, met->u);
03718          read_met_ml2pl(ctl, met, met->v);
03719          read_met_ml2pl(ctl, met, met->w);
03720          read_met_ml2pl(ctl, met, met->h2o);
03721          read_met_ml2pl(ctl, met, met->o3);

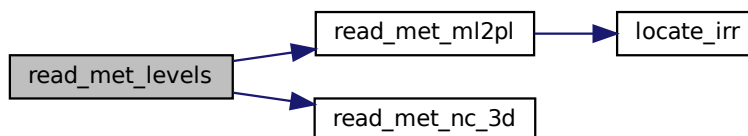
```

```

03722     read_met_ml2pl(ctl, met, met->lwc);
03723     read_met_ml2pl(ctl, met, met->iwc);
03724     if (ctl->vert_vel == 1) {
03725         read_met_ml2pl(ctl, met, met->zeta);
03726         read_met_ml2pl(ctl, met, met->zeta_dot);
03727     }
03728
03729     /* Set new pressure levels... */
03730     met->np = ctl->met_np;
03731     for (int ip = 0; ip < met->np; ip++)
03732         met->p[ip] = ctl->met_p[ip];
03733
03734     /* Create a pressure field... */
03735     for (int i = 0; i < met->nx; i++)
03736         for (int j = 0; j < met->ny; j++)
03737             for (int k = 0; k < met->np; k++) {
03738                 met->patp[i][j][k] = (float) met->p[k];
03739             }
03740 }
03741 } else
03742     ERRMSG("Meteo data format unknown!");
03743
03744 /* Check ordering of pressure levels... */
03745 for (int ip = 1; ip < met->np; ip++)
03746     if (met->p[ip - 1] < met->p[ip])
03747         ERRMSG("Pressure levels must be descending!");
03748 }

```

Here is the call graph for this function:



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

```

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

```

Convert meteo data from model levels to pressure levels.

Definition at line 3752 of file libtrac.c.

```

03755     {
03756
03757     double aux[EP], p[EP];
03758
03759     /* Set timer... */
03760     SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03761     LOG(2, "Interpolate meteo data to pressure levels...");
03762
03763     /* Loop over columns... */
03764     #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03765     for (int ix = 0; ix < met->nx; ix++)
03766         for (int iy = 0; iy < met->ny; iy++) {
03767
03768         /* Copy pressure profile... */
03769         for (int ip = 0; ip < met->np; ip++)
03770             p[ip] = met->p[ix][iy][ip];
03771
03772         /* Interpolate... */
03773         for (int ip = 0; ip < ctl->met_np; ip++) {
03774             double pt = ctl->met_p[ip];

```

```

03775         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03776             pt = p[0];
03777         else if ((pt > p[met->np - 1] && p[1] > p[0])
03778             || (pt < p[met->np - 1] && p[1] < p[0]))
03779             pt = p[met->np - 1];
03780         int ip2 = locate_irr(p, met->np, pt);
03781         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03782             p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03783     }
03784
03785     /* Copy data... */
03786     for (int ip = 0; ip < ctl->met_np; ip++)
03787         var[ix][iy][ip] = (float) aux[ip];
03788 }
03789 }

```

Here is the call graph for this function:



#### 5.21.3.50 read\_met\_nc\_2d() int read\_met\_nc\_2d (

```

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

```

Read and convert 2D variable from meteo data file.

Definition at line 3793 of file libtrac.c.

```

03801     {
03802
03803     char varsel[LEN];
03804
03805     float offset, scalfac;
03806
03807     int varid;
03808
03809     /* Check if variable exists... */
03810     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03811         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03812             WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
03813             return 0;
03814         } else {
03815             sprintf(varsel, "%s", varname2);
03816         } else
03817             sprintf(varsel, "%s", varname);
03818
03819     /* Read packed data... */
03820     if (ctl->met_nc_scale
03821         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03822         && nc_get_att_float(ncid, varid, "scale_factor",
03823             &scalfac) == NC_NOERR) {
03824
03825     /* Allocate... */
03826     short *help;
03827     ALLOC(help, short,

```

```

03828         EX * EY * EP);
03829
03830     /* Read fill value and missing value... */
03831     short fillval, missval;
03832     if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03833         fillval = 0;
03834     if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03835         missval = 0;
03836
03837     /* Write info... */
03838     LOG(2, "Read 2-D variable: %s"
03839         " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03840         varsel, fillval, missval, scalfac, offset);
03841
03842     /* Read data... */
03843     NC(nc_get_var_short(ncid, varid, help));
03844
03845     /* Copy and check data... */
03846 #pragma omp parallel for default(shared) num_threads(12)
03847     for (int ix = 0; ix < met->nx; ix++)
03848         for (int iy = 0; iy < met->ny; iy++) {
03849             if (init)
03850                 dest[ix][iy] = 0;
03851             short aux = help[ARRAY_2D(iy, ix, met->nx)];
03852             if ((fillval == 0 || aux != fillval)
03853                 && (missval == 0 || aux != missval)
03854                 && fabsf(aux * scalfac + offset) < 1e14f)
03855                 dest[ix][iy] += scl * (aux * scalfac + offset);
03856             else
03857                 dest[ix][iy] = GSL_NAN;
03858         }
03859
03860     /* Free... */
03861     free(help);
03862 }
03863
03864 /* Unpacked data... */
03865 else {
03866
03867     /* Allocate... */
03868     float *help;
03869     ALLOC(help, float,
03870         EX * EY);
03871
03872     /* Read fill value and missing value... */
03873     float fillval, missval;
03874     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03875         fillval = 0;
03876     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03877         missval = 0;
03878
03879     /* Write info... */
03880     LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03881         varsel, fillval, missval);
03882
03883     /* Read data... */
03884     NC(nc_get_var_float(ncid, varid, help));
03885
03886     /* Copy and check data... */
03887 #pragma omp parallel for default(shared) num_threads(12)
03888     for (int ix = 0; ix < met->nx; ix++)
03889         for (int iy = 0; iy < met->ny; iy++) {
03890             if (init)
03891                 dest[ix][iy] = 0;
03892             float aux = help[ARRAY_2D(iy, ix, met->nx)];
03893             if ((fillval == 0 || aux != fillval)
03894                 && (missval == 0 || aux != missval)
03895                 && fabsf(aux) < 1e14f)
03896                 dest[ix][iy] += scl * aux;
03897             else
03898                 dest[ix][iy] = GSL_NAN;
03899         }
03900
03901     /* Free... */
03902     free(help);
03903 }
03904
03905 /* Return... */
03906 return 1;
03907 }

```

**5.21.3.51 read\_met\_nc\_3d()** int read\_met\_nc\_3d (  
     int ncid,



```

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

```

Read and convert 3D variable from meteo data file.

Definition at line 3911 of file libtrac.c.

```

3919     {
3920
3921     char varsel[LEN];
3922
3923     float offset, scalfac;
3924
3925     int varid;
3926
3927     /* Check if variable exists... */
3928     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
3929         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
3930             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
3931             return 0;
3932         } else {
3933             sprintf(varsel, "%s", varname2);
3934         } else
3935             sprintf(varsel, "%s", varname);
3936
3937     /* Read packed data... */
3938     if (ctl->met_nc_scale
3939         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
3940         && nc_get_att_float(ncid, varid, "scale_factor",
3941                             &scalfac) == NC_NOERR) {
3942
3943         /* Allocate... */
3944         short *help;
3945         ALLOC(help, short,
3946              EX * EY * EP);
3947
3948         /* Read fill value and missing value... */
3949         short fillval, missval;
3950         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
3951             fillval = 0;
3952         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
3953             missval = 0;
3954
3955         /* Write info... */
3956         LOG(2, "Read 3-D variable: %s "
3957             "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
3958            varsel, fillval, missval, scalfac, offset);
3959
3960         /* Read data... */
3961         NC(nc_get_var_short(ncid, varid, help));
3962
3963         /* Copy and check data... */
3964         #pragma omp parallel for default(shared) num_threads(12)
3965         for (int ix = 0; ix < met->nx; ix++)
3966             for (int iy = 0; iy < met->ny; iy++)
3967                 for (int ip = 0; ip < met->np; ip++) {
3968                     if (init)
3969                         dest[ix][iy][ip] = 0;
3970                     short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
3971                     if ((fillval == 0 || aux != fillval)
3972                         && (missval == 0 || aux != missval)
3973                         && fabsf(aux * scalfac + offset) < 1e14f)
3974                         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
3975                     else
3976                         dest[ix][iy][ip] = GSL_NAN;
3977                 }
3978
3979         /* Free... */
3980         free(help);
3981     }
3982
3983     /* Unpacked data... */
3984     else {
3985
3986         /* Allocate... */
3987         float *help;
3988         ALLOC(help, float,
3989              EX * EY * EP);

```

```

03990
03991      /* Read fill value and missing value... */
03992      float fillval, missval;
03993      if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03994          fillval = 0;
03995      if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03996          missval = 0;
03997
03998      /* Write info... */
03999      LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
04000          varsel, fillval, missval);
04001
04002      /* Read data... */
04003      NC(nc_get_var_float(ncid, varid, help));
04004
04005      /* Copy and check data... */
04006      #pragma omp parallel for default(shared) num_threads(12)
04007      for (int ix = 0; ix < met->nx; ix++)
04008          for (int iy = 0; iy < met->ny; iy++)
04009              for (int ip = 0; ip < met->np; ip++) {
04010                  if (init)
04011                      dest[ix][iy][ip] = 0;
04012                  float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04013                  if ((fillval == 0 || aux != fillval)
04014                      && (missval == 0 || aux != missval)
04015                      && fabsf(aux) < 1e14f)
04016                      dest[ix][iy][ip] += scl * aux;
04017                  else
04018                      dest[ix][iy][ip] = GSL_NAN;
04019              }
04020
04021      /* Free... */
04022      free(help);
04023  }
04024
04025      /* Return... */
04026      return 1;
04027  }

```

**5.21.3.52 read\_met\_pbl()** void read\_met\_pbl (  
     met\_t \* met )

Calculate pressure of the boundary layer.

Definition at line 4031 of file libtrac.c.

```

04032      {
04033
04034      /* Set timer... */
04035      SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
04036      LOG(2, "Calculate planetary boundary layer...");
04037
04038      /* Parameters used to estimate the height of the PBL
04039       (e.g., Voegelezang and Holtslag, 1996; Seidel et al., 2012)... */
04040      const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
04041
04042      /* Loop over grid points... */
04043      #pragma omp parallel for default(shared) collapse(2)
04044      for (int ix = 0; ix < met->nx; ix++)
04045          for (int iy = 0; iy < met->ny; iy++) {
04046
04047              /* Set bottom level of PBL... */
04048              double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
04049
04050              /* Find lowest level near the bottom... */
04051              int ip;
04052              for (ip = 1; ip < met->np; ip++)
04053                  if (met->p[ip] < pbl_bot)
04054                      break;
04055
04056              /* Get near surface data... */
04057              double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
04058                             met->p[ip], met->z[ix][iy][ip], pbl_bot);
04059              double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
04060                             met->p[ip], met->t[ix][iy][ip], pbl_bot);
04061              double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
04062                             met->p[ip], met->u[ix][iy][ip], pbl_bot);
04063              double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
04064                             met->p[ip], met->v[ix][iy][ip], pbl_bot);
04065              double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],

```

```

04066         met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
04067     double tvs = THETAVIRT(pbl_bot, ts, h2os);
04068
04069     /* Init... */
04070     double rib_old = 0;
04071
04072     /* Loop over levels... */
04073     for (; ip < met->np; ip++) {
04074
04075         /* Get squared horizontal wind speed... */
04076         double vh2
04077             = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
04078         vh2 = GSL_MAX(vh2, SQR(umin));
04079
04080         /* Calculate bulk Richardson number... */
04081         double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
04082             * (THETAVIRT(met->p[ip], met->t[ix][iy][ip],
04083                 met->h2o[ix][iy][ip]) - tvs) / vh2;
04084
04085         /* Check for critical value... */
04086         if (rib >= rib_crit) {
04087             met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
04088                 rib, met->p[ip], rib_crit));
04089             if (met->pbl[ix][iy] > pbl_bot)
04090                 met->pbl[ix][iy] = (float) pbl_bot;
04091             break;
04092         }
04093
04094         /* Save Richardson number... */
04095         rib_old = rib;
04096     }
04097 }
04098 }

```

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

Create meteo data with periodic boundary conditions.

Definition at line 4102 of file libtrac.c.

```

04103     {
04104
04105         /* Set timer... */
04106         SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
04107         LOG(2, "Apply periodic boundary conditions...");
04108
04109         /* Check longitudes... */
04110         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
04111             + met->lon[1] - met->lon[0] - 360) < 0.01))
04112             return;
04113
04114         /* Increase longitude counter... */
04115         if ((++met->nx) > EX)
04116             ERRMSG("Cannot create periodic boundary conditions!");
04117
04118         /* Set longitude... */
04119         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
04120
04121         /* Loop over latitudes and pressure levels... */
04122         #pragma omp parallel for default(shared)
04123         for (int iy = 0; iy < met->ny; iy++) {
04124             met->ps[met->nx - 1][iy] = met->ps[0][iy];
04125             met->zs[met->nx - 1][iy] = met->zs[0][iy];
04126             met->ts[met->nx - 1][iy] = met->ts[0][iy];
04127             met->us[met->nx - 1][iy] = met->us[0][iy];
04128             met->vs[met->nx - 1][iy] = met->vs[0][iy];
04129             for (int ip = 0; ip < met->np; ip++) {
04130                 met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
04131                 met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
04132                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
04133                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
04134                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
04135                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
04136                 met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
04137                 met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
04138             }
04139         }
04140     }

```

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

Calculate potential vorticity.

Definition at line 4144 of file libtrac.c.

```

04145     {
04146
04147     double pows[EP];
04148
04149     /* Set timer... */
04150     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
04151     LOG(2, "Calculate potential vorticity...");
04152
04153     /* Set powers... */
04154     #pragma omp parallel for default(shared)
04155     for (int ip = 0; ip < met->np; ip++)
04156         pows[ip] = pow(1000. / met->p[ip], 0.286);
04157
04158     /* Loop over grid points... */
04159     #pragma omp parallel for default(shared)
04160     for (int ix = 0; ix < met->nx; ix++) {
04161
04162         /* Set indices... */
04163         int ix0 = GSL_MAX(ix - 1, 0);
04164         int ix1 = GSL_MIN(ix + 1, met->nx - 1);
04165
04166         /* Loop over grid points... */
04167         for (int iy = 0; iy < met->ny; iy++) {
04168
04169             /* Set indices... */
04170             int iy0 = GSL_MAX(iy - 1, 0);
04171             int iy1 = GSL_MIN(iy + 1, met->ny - 1);
04172
04173             /* Set auxiliary variables... */
04174             double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
04175             double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
04176             double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
04177             double c0 = cos(met->lat[iy0] / 180. * M_PI);
04178             double c1 = cos(met->lat[iy1] / 180. * M_PI);
04179             double cr = cos(latr / 180. * M_PI);
04180             double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
04181
04182             /* Loop over grid points... */
04183             for (int ip = 0; ip < met->np; ip++) {
04184
04185                 /* Get gradients in longitude... */
04186                 double dtdx
04187                     = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
04188                 double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
04189
04190                 /* Get gradients in latitude... */
04191                 double dtdy
04192                     = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
04193                 double dudx
04194                     = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
04195
04196                 /* Set indices... */
04197                 int ip0 = GSL_MAX(ip - 1, 0);
04198                 int ip1 = GSL_MIN(ip + 1, met->np - 1);
04199
04200                 /* Get gradients in pressure... */
04201                 double dtdp, dudp, dvdp;
04202                 double dp0 = 100. * (met->p[ip] - met->p[ip0]);
04203                 double dp1 = 100. * (met->p[ip1] - met->p[ip]);
04204                 if (ip != ip0 && ip != ip1) {
04205                     double denom = dp0 * dp1 * (dp0 + dp1);
04206                     dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
04207                         - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
04208                         + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
04209                         / denom;
04210                     dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
04211                         - dp1 * dp1 * met->u[ix][iy][ip0]
04212                         + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
04213                         / denom;
04214                     dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
04215                         - dp1 * dp1 * met->v[ix][iy][ip0]
04216                         + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
04217                         / denom;
04218                 } else {
04219                     double denom = dp0 + dp1;
04220                     dtdp =
04221                         (met->t[ix][iy][ip1] * pows[ip1] -
04222                         met->t[ix][iy][ip0] * pows[ip0]) / denom;
04223                     dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;

```

```

04224         dvdp = (met->v[ix][iy][ipl] - met->v[ix][iy][ip0]) / denom;
04225     }
04226
04227     /* Calculate PV... */
04228     met->pv[ix][iy][ip] = (float)
04229         (le6 * G0 *
04230          (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
04231     }
04232 }
04233 }
04234
04235 /* Fix for polar regions... */
04236 #pragma omp parallel for default(shared)
04237 for (int ix = 0; ix < met->nx; ix++)
04238     for (int ip = 0; ip < met->np; ip++) {
04239         met->pv[ix][0][ip]
04240             = met->pv[ix][1][ip]
04241             = met->pv[ix][2][ip];
04242         met->pv[ix][met->ny - 1][ip]
04243             = met->pv[ix][met->ny - 2][ip]
04244             = met->pv[ix][met->ny - 3][ip];
04245     }
04246 }

```

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

Downsampling of meteo data.

Definition at line 4250 of file libtrac.c.

```

04252     {
04253
04254         met_t *help;
04255
04256         /* Check parameters... */
04257         if (ctl->met_dx <= 1 && ctl->met_dy <= 1 && ctl->met_dx <= 1 &&
04258             && ctl->met_sy <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
04259             return;
04260
04261         /* Set timer... */
04262         SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
04263         LOG(2, "Downsampling of meteo data...");
04264
04265         /* Allocate... */
04266         ALLOC(help, met_t, 1);
04267
04268         /* Copy data... */
04269         help->nx = met->nx;
04270         help->ny = met->ny;
04271         help->np = met->np;
04272         memcpy(help->lon, met->lon, sizeof(met->lon));
04273         memcpy(help->lat, met->lat, sizeof(met->lat));
04274         memcpy(help->p, met->p, sizeof(met->p));
04275
04276         /* Smoothing... */
04277         for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
04278             for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
04279                 for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
04280                     help->ps[ix][iy] = 0;
04281                     help->zs[ix][iy] = 0;
04282                     help->ts[ix][iy] = 0;
04283                     help->us[ix][iy] = 0;
04284                     help->vs[ix][iy] = 0;
04285                     help->t[ix][iy][ip] = 0;
04286                     help->u[ix][iy][ip] = 0;
04287                     help->v[ix][iy][ip] = 0;
04288                     help->w[ix][iy][ip] = 0;
04289                     help->h2o[ix][iy][ip] = 0;
04290                     help->o3[ix][iy][ip] = 0;
04291                     help->lwc[ix][iy][ip] = 0;
04292                     help->iwc[ix][iy][ip] = 0;
04293                     float wsum = 0;
04294                     for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
04295                         ix2++) {
04296                         int ix3 = ix2;
04297                         if (ix3 < 0)
04298                             ix3 += met->nx;

```

```

04299     else if (ix3 >= met->nx)
04300         ix3 -= met->nx;
04301
04302     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
04303          iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
04304         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
04305              ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
04306             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
04307                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
04308                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
04309             help->ps[ix][iy] += w * met->ps[ix3][iy2];
04310             help->zs[ix][iy] += w * met->zs[ix3][iy2];
04311             help->ts[ix][iy] += w * met->ts[ix3][iy2];
04312             help->us[ix][iy] += w * met->us[ix3][iy2];
04313             help->vs[ix][iy] += w * met->vs[ix3][iy2];
04314             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
04315             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
04316             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
04317             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
04318             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
04319             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
04320             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
04321             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
04322             wsum += w;
04323         }
04324     }
04325     help->ps[ix][iy] /= wsum;
04326     help->zs[ix][iy] /= wsum;
04327     help->ts[ix][iy] /= wsum;
04328     help->us[ix][iy] /= wsum;
04329     help->vs[ix][iy] /= wsum;
04330     help->t[ix][iy][ip] /= wsum;
04331     help->u[ix][iy][ip] /= wsum;
04332     help->v[ix][iy][ip] /= wsum;
04333     help->w[ix][iy][ip] /= wsum;
04334     help->h2o[ix][iy][ip] /= wsum;
04335     help->o3[ix][iy][ip] /= wsum;
04336     help->lwc[ix][iy][ip] /= wsum;
04337     help->iwc[ix][iy][ip] /= wsum;
04338 }
04339 }
04340 }
04341
04342 /* Downsampling... */
04343 met->nx = 0;
04344 for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04345     met->lon[met->nx] = help->lon[ix];
04346     met->ny = 0;
04347     for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04348         met->lat[met->ny] = help->lat[iy];
04349         met->ps[met->nx][met->ny] = help->ps[ix][iy];
04350         met->zs[met->nx][met->ny] = help->zs[ix][iy];
04351         met->ts[met->nx][met->ny] = help->ts[ix][iy];
04352         met->us[met->nx][met->ny] = help->us[ix][iy];
04353         met->vs[met->nx][met->ny] = help->vs[ix][iy];
04354         met->np = 0;
04355         for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04356             met->p[met->np] = help->p[ip];
04357             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04358             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04359             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04360             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04361             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04362             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04363             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
04364             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04365             met->np++;
04366         }
04367         met->ny++;
04368     }
04369     met->nx++;
04370 }
04371
04372 /* Free... */
04373 free(help);
04374 }

```

```

5.21.3.56 read_met_surface() void read_met_surface (
    int ncid,
    met_t * met,
    ctl_t * ctl )

```

Read surface data.

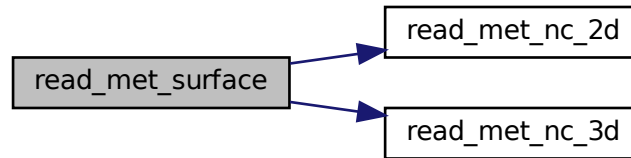
Definition at line 4378 of file libtrac.c.

```

04381     {
04382
04383     /* Set timer... */
04384     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04385     LOG(2, "Read surface data...");
04386
04387     /* MPTRAC meteo data... */
04388     if (ctl->clams_met_data == 0) {
04389
04390         /* Read surface pressure... */
04391         if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
04392             if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04393                 WARN("Cannot not read surface pressure data (use lowest level!)");
04394                 for (int ix = 0; ix < met->nx; ix++)
04395                     for (int iy = 0; iy < met->ny; iy++)
04396                         met->ps[ix][iy] = (float) met->p[0];
04397             }
04398         } else
04399             for (int ix = 0; ix < met->nx; ix++)
04400                 for (int iy = 0; iy < met->ny; iy++)
04401                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04402
04403         /* Read geopotential height at the surface... */
04404         if (!read_met_nc_2d(ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
04405             if (!read_met_nc_2d(ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
04406                 WARN("Cannot read surface geopotential height!");
04407
04408         /* Read temperature at the surface... */
04409         if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
04410             WARN("Cannot read surface temperature!");
04411
04412         /* Read zonal wind at the surface... */
04413         if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
04414             WARN("Cannot read surface zonal wind!");
04415
04416         /* Read meridional wind at the surface... */
04417         if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
04418             WARN("Cannot read surface meridional wind!");
04419     }
04420
04421     /* CLaMS meteo data... */
04422     else {
04423
04424         /* Read surface pressure... */
04425         if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04426             WARN("Cannot not read surface pressure data (use lowest level!)");
04427             for (int ix = 0; ix < met->nx; ix++)
04428                 for (int iy = 0; iy < met->ny; iy++)
04429                     met->ps[ix][iy] = (float) met->p[0];
04430         }
04431
04432         /* Read geopotential height at the surface
04433            (use lowermost level of 3-D data field)... */
04434         float *help;
04435         ALLOC(help, float,
04436              EX * EY * EP);
04437         memcpy(help, met->p1, sizeof(met->p1));
04438         if (!read_met_nc_3d(ncid, "gph", "GPH", ctl, met, met->p1, (float) (1e-3 / G0), 1)) {
04439             ERRMSG("Cannot read geopotential height!");
04440         } else
04441             for (int ix = 0; ix < met->nx; ix++)
04442                 for (int iy = 0; iy < met->ny; iy++)
04443                     met->zs[ix][iy] = met->p1[ix][iy][0];
04444         memcpy(met->p1, help, sizeof(met->p1));
04445         free(help);
04446
04447         /* Read temperature at the surface... */
04448         if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
04449             WARN("Cannot read surface temperature!");
04450
04451         /* Read zonal wind at the surface... */
04452         if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
04453             WARN("Cannot read surface zonal wind!");
04454
04455         /* Read meridional wind at the surface... */
04456         if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
04457             WARN("Cannot read surface meridional wind!");
04458     }
04459 }
04460
04461 }
04462 }

```

Here is the call graph for this function:



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

Calculate tropopause data.

Definition at line 4466 of file libtrac.c.

```

04469     {
04470
04471     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04472           th2[200], z[EP], z2[200];
04473
04474     /* Set timer... */
04475     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04476     LOG(2, "Calculate tropopause...");
04477
04478     /* Get altitude and pressure profiles... */
04479     #pragma omp parallel for default(shared)
04480     for (int iz = 0; iz < met->np; iz++)
04481         z[iz] = Z(met->p[iz]);
04482     #pragma omp parallel for default(shared)
04483     for (int iz = 0; iz <= 190; iz++) {
04484         z2[iz] = 4.5 + 0.1 * iz;
04485         p2[iz] = P(z2[iz]);
04486     }
04487
04488     /* Do not calculate tropopause... */
04489     if (ctl->met_tropo == 0)
04490     #pragma omp parallel for default(shared) collapse(2)
04491     for (int ix = 0; ix < met->nx; ix++)
04492     for (int iy = 0; iy < met->ny; iy++)
04493         met->pt[ix][iy] = GSL_NAN;
04494
04495     /* Use tropopause climatology... */
04496     else if (ctl->met_tropo == 1) {
04497     #pragma omp parallel for default(shared) collapse(2)
04498     for (int ix = 0; ix < met->nx; ix++)
04499     for (int iy = 0; iy < met->ny; iy++)
04500         met->pt[ix][iy] = (float) clim_tropo(clim, met->time, met->lat[iy]);
04501     }
04502
04503     /* Use cold point... */
04504     else if (ctl->met_tropo == 2) {
04505
04506         /* Loop over grid points... */
04507     #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04508     for (int ix = 0; ix < met->nx; ix++)
04509     for (int iy = 0; iy < met->ny; iy++) {
04510
04511         /* Interpolate temperature profile... */
04512         for (int iz = 0; iz < met->np; iz++)
04513             t[iz] = met->t[ix][iy][iz];
04514         spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
  
```



```

04515
04516     /* Find minimum... */
04517     int iz = (int) gsl_stats_min_index(t2, 1, 171);
04518     if (iz > 0 && iz < 170)
04519         met->pt[ix][iy] = (float) p2[iz];
04520     else
04521         met->pt[ix][iy] = GSL_NAN;
04522 }
04523 }
04524
04525 /* Use WMO definition... */
04526 else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04527
04528     /* Loop over grid points... */
04529     #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04530     for (int ix = 0; ix < met->nx; ix++)
04531         for (int iy = 0; iy < met->ny; iy++) {
04532
04533             /* Interpolate temperature profile... */
04534             int iz;
04535             for (iz = 0; iz < met->np; iz++)
04536                 t[iz] = met->t[ix][iy][iz];
04537             spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04538
04539             /* Find 1st tropopause... */
04540             met->pt[ix][iy] = GSL_NAN;
04541             for (iz = 0; iz <= 170; iz++) {
04542                 int found = 1;
04543                 for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04544                     if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04545                         ctl->met_tropo_lapse) {
04546                         found = 0;
04547                         break;
04548                     }
04549                 if (found) {
04550                     if (iz > 0 && iz < 170)
04551                         met->pt[ix][iy] = (float) p2[iz];
04552                     break;
04553                 }
04554             }
04555
04556             /* Find 2nd tropopause... */
04557             if (ctl->met_tropo == 4) {
04558                 met->pt[ix][iy] = GSL_NAN;
04559                 for (; iz <= 170; iz++) {
04560                     int found = 1;
04561                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04562                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04563                             ctl->met_tropo_lapse_sep) {
04564                             found = 0;
04565                             break;
04566                         }
04567                     if (found)
04568                         break;
04569                 }
04570                 for (; iz <= 170; iz++) {
04571                     int found = 1;
04572                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04573                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04574                             ctl->met_tropo_lapse) {
04575                             found = 0;
04576                             break;
04577                         }
04578                     if (found) {
04579                         if (iz > 0 && iz < 170)
04580                             met->pt[ix][iy] = (float) p2[iz];
04581                         break;
04582                     }
04583                 }
04584             }
04585         }
04586     }
04587
04588     /* Use dynamical tropopause... */
04589     else if (ctl->met_tropo == 5) {
04590
04591         /* Loop over grid points... */
04592         #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04593         for (int ix = 0; ix < met->nx; ix++)
04594             for (int iy = 0; iy < met->ny; iy++) {
04595
04596                 /* Interpolate potential vorticity profile... */
04597                 for (int iz = 0; iz < met->np; iz++)
04598                     pv[iz] = met->pv[ix][iy][iz];
04599                 spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04600
04601                 /* Interpolate potential temperature profile... */

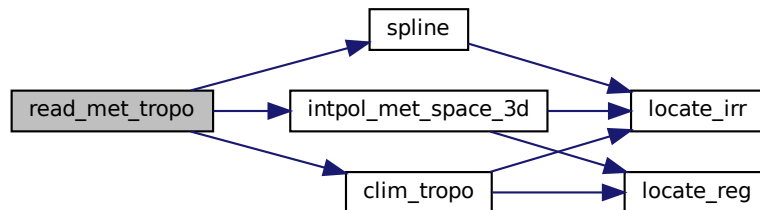
```

```

04602     for (int iz = 0; iz < met->np; iz++)
04603         th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04604     spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04605
04606     /* Find dynamical tropopause... */
04607     met->pt[ix][iy] = GSL_NAN;
04608     for (int iz = 0; iz <= 170; iz++)
04609         if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04610             || th2[iz] >= ctl->met_tropo_theta) {
04611             if (iz > 0 && iz < 170)
04612                 met->pt[ix][iy] = (float) p2[iz];
04613             break;
04614         }
04615     }
04616 }
04617
04618 else
04619     ERRMSG("Cannot calculate tropopause!");
04620
04621 /* Interpolate temperature, geopotential height, and water vapor vmr... */
04622 #pragma omp parallel for default(shared) collapse(2)
04623 for (int ix = 0; ix < met->nx; ix++)
04624     for (int iy = 0; iy < met->ny; iy++) {
04625         double h2ot, tt, zt;
04626         INTPOL_INIT;
04627         intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04628                             met->lat[iy], &tt, ci, cw, 1);
04629         intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04630                             met->lat[iy], &zt, ci, cw, 0);
04631         intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04632                             met->lat[iy], &h2ot, ci, cw, 0);
04633         met->tt[ix][iy] = (float) tt;
04634         met->zt[ix][iy] = (float) zt;
04635         met->h2ot[ix][iy] = (float) h2ot;
04636     }
04637 }

```

Here is the call graph for this function:



**5.21.3.58 read\_obs()** void read\_obs (

```

    char * filename,
    double * rt,
    double * rz,
    double * rlon,
    double * rlat,
    double * robs,
    int * nobss )

```

Read observation data.

Definition at line 4641 of file libtrac.c.

```

04648     {
04649

```

```

04650 FILE *in;
04651
04652 char line[LEN];
04653
04654 /* Open observation data file... */
04655 LOG(1, "Read observation data: %s", filename);
04656 if (!(in = fopen(filename, "r")))
04657     ERRMSG("Cannot open file!");
04658
04659 /* Read observations... */
04660 while (fgets(line, LEN, in))
04661     if (sscanf(line, "%lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
04662               &rln[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
04663         if ((++(*nobs)) >= NOBS)
04664             ERRMSG("Too many observations!");
04665
04666 /* Close observation data file... */
04667 fclose(in);
04668
04669 /* Check time... */
04670 for (int i = 1; i < *nobs; i++)
04671     if (rt[i] < rt[i - 1])
04672         ERRMSG("Time must be ascending!");
04673
04674 /* Write info... */
04675 int n = *nobs;
04676 double mini, maxi;
04677 LOG(2, "Number of observations: %d", *nobs);
04678 gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
04679 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04680 gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
04681 LOG(2, "Altitude range: %g ... %g km", mini, maxi);
04682 gsl_stats_minmax(&mini, &maxi, rln, 1, (size_t) n);
04683 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04684 gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
04685 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04686 gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
04687 LOG(2, "Observation range: %g ... %g", mini, maxi);
04688 }

```

**5.21.3.59 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 4692 of file libtrac.c.

```

04699 {
04700
04701 FILE *in = NULL;
04702
04703 char fullname1[LEN], fullname2[LEN], rval[LEN];
04704
04705 int contain = 0, i;
04706
04707 /* Open file... */
04708 if (filename[strlen(filename) - 1] != '-')
04709     if (!(in = fopen(filename, "r")))
04710         ERRMSG("Cannot open file!");
04711
04712 /* Set full variable name... */
04713 if (arridx >= 0) {
04714     sprintf(fullname1, "%s[%d]", varname, arridx);
04715     sprintf(fullname2, "%s[*]", varname);
04716 } else {
04717     sprintf(fullname1, "%s", varname);
04718     sprintf(fullname2, "%s", varname);
04719 }
04720
04721 /* Read data... */
04722 if (in != NULL) {

```

```

04723     char dummy[LEN], line[LEN], rvarname[LEN];
04724     while (fgets(line, LEN, in)) {
04725         if (sscanf(line, "%4999s %4999s", rvarname, dummy, rval) == 3)
04726             if (strcasestr(rvarname, fullname1) == 0 ||
04727                 strcasestr(rvarname, fullname2) == 0) {
04728                 contain = 1;
04729                 break;
04730             }
04731     }
04732 }
04733 for (i = 1; i < argc - 1; i++)
04734     if (strcasestr(argv[i], fullname1) == 0 ||
04735         strcasestr(argv[i], fullname2) == 0) {
04736         sprintf(rval, "%s", argv[i + 1]);
04737         contain = 1;
04738         break;
04739     }
04740
04741 /* Close file... */
04742 if (in != NULL)
04743     fclose(in);
04744
04745 /* Check for missing variables... */
04746 if (!contain) {
04747     if (strlen(defvalue) > 0)
04748         sprintf(rval, "%s", defvalue);
04749     else
04750         ERRMSG("Missing variable %s!\n", fullname1);
04751 }
04752
04753 /* Write info... */
04754 LOG(1, "%s = %s", fullname1, rval);
04755
04756 /* Return values... */
04757 if (value != NULL)
04758     sprintf(value, "%s", rval);
04759 return atof(rval);
04760 }

```

**5.21.3.60 sedi()** double sedi (  
     double p,  
     double T,  
     double rp,  
     double rhop )

Calculate sedimentation velocity.

Definition at line 4764 of file libtrac.c.

```

04768     {
04769
04770     /* Convert particle radius from microns to m... */
04771     rp *= 1e-6;
04772
04773     /* Density of dry air [kg / m^3]... */
04774     double rho = RHO(p, T);
04775
04776     /* Dynamic viscosity of air [kg / (m s)]... */
04777     double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04778
04779     /* Thermal velocity of an air molecule [m / s]... */
04780     double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04781
04782     /* Mean free path of an air molecule [m]... */
04783     double lambda = 2. * eta / (rho * v);
04784
04785     /* Knudsen number for air (dimensionless)... */
04786     double K = lambda / rp;
04787
04788     /* Cunningham slip-flow correction (dimensionless)... */
04789     double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04790
04791     /* Sedimentation velocity [m / s]... */
04792     return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
04793 }

```

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

Spline interpolation.

Definition at line 4797 of file [libtrac.c](#).

```

04804     {
04805
04806     /* Cubic spline interpolation... */
04807     if (method == 1) {
04808
04809     /* Allocate... */
04810     gsl_interp_accel *acc;
04811     gsl_spline *s;
04812     acc = gsl_interp_accel_alloc();
04813     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04814
04815     /* Interpolate profile... */
04816     gsl_spline_init(s, x, y, (size_t) n);
04817     for (int i = 0; i < n2; i++)
04818         if (x2[i] <= x[0])
04819             y2[i] = y[0];
04820         else if (x2[i] >= x[n - 1])
04821             y2[i] = y[n - 1];
04822         else
04823             y2[i] = gsl_spline_eval(s, x2[i], acc);
04824
04825     /* Free... */
04826     gsl_spline_free(s);
04827     gsl_interp_accel_free(acc);
04828     }
04829
04830     /* Linear interpolation... */
04831     else {
04832     for (int i = 0; i < n2; i++)
04833         if (x2[i] <= x[0])
04834             y2[i] = y[0];
04835         else if (x2[i] >= x[n - 1])
04836             y2[i] = y[n - 1];
04837         else {
04838             int idx = locate_irr(x, n, x2[i]);
04839             y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04840         }
04841     }
04842 }
```

Here is the call graph for this function:



**5.21.3.62 stddev()** float stddev (  
float \* data,  
int n )

Calculate standard deviation.

Definition at line 4846 of file libtrac.c.

```

04848     {
04849
04850     if (n <= 0)
04851         return 0;
04852
04853     float mean = 0, var = 0;
04854
04855     for (int i = 0; i < n; ++i) {
04856         mean += data[i];
04857         var += SQR(data[i]);
04858     }
04859
04860     var = var / (float) n - SQR(mean / (float) n);
04861
04862     return (var > 0 ? sqrtf(var) : 0);
04863 }
```

**5.21.3.63** **sza()** double sza (  
     double sec,  
     double lon,  
     double lat )

Calculate solar zenith angle.

Definition at line 4867 of file libtrac.c.

```

04870     {
04871
04872     double D, dec, e, g, GMST, h, L, LST, q, ra;
04873
04874     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
04875     D = sec / 86400 - 0.5;
04876
04877     /* Geocentric apparent ecliptic longitude [rad]... */
04878     g = (357.529 + 0.98560028 * D) * M_PI / 180;
04879     q = 280.459 + 0.98564736 * D;
04880     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
04881
04882     /* Mean obliquity of the ecliptic [rad]... */
04883     e = (23.439 - 0.00000036 * D) * M_PI / 180;
04884
04885     /* Declination [rad]... */
04886     dec = asin(sin(e) * sin(L));
04887
04888     /* Right ascension [rad]... */
04889     ra = atan2(cos(e) * sin(L), cos(L));
04890
04891     /* Greenwich Mean Sidereal Time [h]... */
04892     GMST = 18.697374558 + 24.06570982441908 * D;
04893
04894     /* Local Sidereal Time [h]... */
04895     LST = GMST + lon / 15;
04896
04897     /* Hour angle [rad]... */
04898     h = LST / 12 * M_PI - ra;
04899
04900     /* Convert latitude... */
04901     lat *= M_PI / 180;
04902
04903     /* Return solar zenith angle [rad]... */
04904     return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
04905 }
```

**5.21.3.64 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 4909 of file libtrac.c.

```

04917     {
04918
04919     struct tm t0, t1;
04920
04921     t0.tm_year = 100;
04922     t0.tm_mon = 0;
04923     t0.tm_mday = 1;
04924     t0.tm_hour = 0;
04925     t0.tm_min = 0;
04926     t0.tm_sec = 0;
04927
04928     t1.tm_year = year - 1900;
04929     t1.tm_mon = mon - 1;
04930     t1.tm_mday = day;
04931     t1.tm_hour = hour;
04932     t1.tm_min = min;
04933     t1.tm_sec = sec;
04934
04935     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04936 }
```

**5.21.3.65 timer()** void timer (

```

    const char * name,
    const char * group,
    int output )
```

Measure wall-clock time.

Definition at line 4940 of file libtrac.c.

```

04943     {
04944
04945     static char names[NTIMER][100], groups[NTIMER][100];
04946
04947     static double rt_name[NTIMER], rt_group[NTIMER],
04948         rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;
04949
04950     static int iname = -1, igrup = -1, nname, ngroup, ct_name[NTIMER];
04951
04952     /* Get time... */
04953     t1 = omp_get_wtime();
04954     dt = t1 - t0;
04955
04956     /* Add elapsed time to current timers... */
04957     if (iname >= 0) {
04958         rt_name[iname] += dt;
04959         rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
04960         rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
04961         ct_name[iname]++;
04962     }
04963     if (igrup >= 0)
04964         rt_group[igrup] += t1 - t0;
04965
04966     /* Report timers... */
04967     if (output) {
04968         for (int i = 0; i < nname; i++)
04969             LOG(1, "TIMER_%s = %.3f s (min= %g s, mean= %g s, "
04970                 " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
04971                 rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
04972         for (int i = 0; i < ngroup; i++)
```

```

04973     LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
04974     double total = 0.0;
04975     for (int i = 0; i < nname; i++)
04976         total += rt_name[i];
04977     LOG(1, "TIMER_TOTAL = %.3f s", total);
04978 }
04979
04980 /* Identify IDs of next timer... */
04981 for (iname = 0; iname < nname; iname++)
04982     if (strcasecmp(name, names[iname]) == 0)
04983         break;
04984 for (igroup = 0; igroup < ngroup; igroup++)
04985     if (strcasecmp(group, groups[igroup]) == 0)
04986         break;
04987
04988 /* Check whether this is a new timer... */
04989 if (iname >= nname) {
04990     sprintf(names[iname], "%s", name);
04991     if ((++nname) > NTIMER)
04992         ERRMSG("Too many timers!");
04993 }
04994
04995 /* Check whether this is a new group... */
04996 if (igroup >= ngroup) {
04997     sprintf(groups[igroup], "%s", group);
04998     if ((++ngroup) > NTIMER)
04999         ERRMSG("Too many groups!");
05000 }
05001
05002 /* Save starting time... */
05003 t0 = t1;
05004 }

```

**5.21.3.66 tropo\_weight()** double tropo\_weight (  
     clim\_t \* clim,  
     double t,  
     double lat,  
     double p )

Get weighting factor based on tropopause distance.

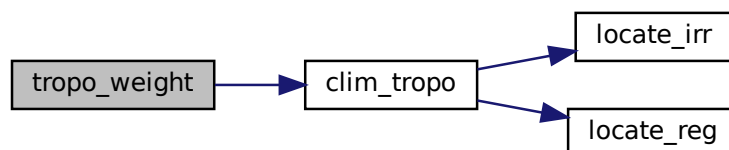
Definition at line 5008 of file libtrac.c.

```

05012     {
05013
05014     /* Get tropopause pressure... */
05015     double pt = clim_tropo(clim, t, lat);
05016
05017     /* Get pressure range... */
05018     double p1 = pt * 0.866877899;
05019     double p0 = pt / 0.866877899;
05020
05021     /* Get weighting factor... */
05022     if (p > p0)
05023         return 1;
05024     else if (p < p1)
05025         return 0;
05026     else
05027         return LIN(p0, 1.0, p1, 0.0, p);
05028 }

```

Here is the call graph for this function:





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

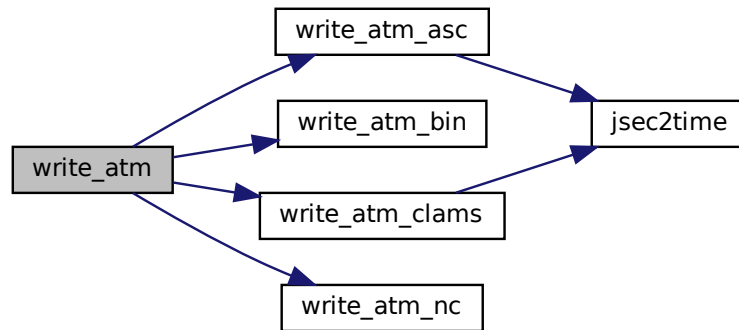
Write atmospheric data.

Definition at line 5032 of file libtrac.c.

```

05036     {
05037
05038     /* Set timer... */
05039     SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
05040
05041     /* Write info... */
05042     LOG(1, "Write atmospheric data: %s", filename);
05043
05044     /* Write ASCII data... */
05045     if (ctl->atm_type == 0)
05046         write_atm_asc(filename, ctl, atm, t);
05047
05048     /* Write binary data... */
05049     else if (ctl->atm_type == 1)
05050         write_atm_bin(filename, ctl, atm);
05051
05052     /* Write netCDF data... */
05053     else if (ctl->atm_type == 2)
05054         write_atm_nc(filename, ctl, atm);
05055
05056     /* Write CLaMS data... */
05057     else if (ctl->atm_type == 3)
05058         write_atm_clams(ctl, atm, t);
05059
05060     /* Error... */
05061     else
05062         ERRMSG("Atmospheric data type not supported!");
05063
05064     /* Write info... */
05065     double mini, maxi;
05066     LOG(2, "Number of particles: %d", atm->np);
05067     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
05068     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
05069     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
05070     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
05071     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
05072     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
05073     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
05074     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
05075     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
05076     for (int iq = 0; iq < ctl->nq; iq++) {
05077         char msg[LEN];
05078         sprintf(msg, "Quantity %s range: %s ... %s %s",
05079                 ctl->qnt_name[iq], ctl->qnt_format[iq],
05080                 ctl->qnt_format[iq], ctl->qnt_unit[iq]);
05081         gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
05082         LOG(2, msg, mini, maxi);
05083     }
05084 }
```

Here is the call graph for this function:



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

Write atmospheric data in ASCII format.

Definition at line 5088 of file libtrac.c.

```

05092     {
05093
05094     FILE *out;
05095
05096     /* Set time interval for output... */
05097     double t0 = t - 0.5 * ctl->dt_mod;
05098     double t1 = t + 0.5 * ctl->dt_mod;
05099
05100     /* Check if gnuplot output is requested... */
05101     if (ctl->atm_gpfile[0] != '-') {
05102
05103         /* Create gnuplot pipe... */
05104         if (!(out = popen("gnuplot", "w")))
05105             ERRMSG("Cannot create pipe to gnuplot!");
05106
05107         /* Set plot filename... */
05108         fprintf(out, "set out \"%s.png\"\n", filename);
05109
05110         /* Set time string... */
05111         double r;
05112         int year, mon, day, hour, min, sec;
05113         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05114         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05115             year, mon, day, hour, min);
05116
05117         /* Dump gnuplot file to pipe... */
05118         FILE *in;
05119         if (!(in = fopen(ctl->atm_gpfile, "r")))
05120             ERRMSG("Cannot open file!");
05121         char line[LEN];
05122         while (fgets(line, LEN, in))
05123             fprintf(out, "%s", line);
05124         fclose(in);
05125     }
05126
05127     else {
05128
05129         /* Create file... */

```

```

05130     if (!(out = fopen(filename, "w")))
05131     ERRMSG("Cannot create file!");
05132 }
05133
05134 /* Write header... */
05135 fprintf(out,
05136         "# $1 = time [s]\n"
05137         "# $2 = altitude [km]\n"
05138         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05139 for (int iq = 0; iq < ctl->nq; iq++)
05140     fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
05141             ctl->qnt_unit[iq]);
05142 fprintf(out, "\n");
05143
05144 /* Write data... */
05145 for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
05146
05147     /* Check time... */
05148     if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05149         continue;
05150
05151     /* Write output... */
05152     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
05153             atm->lon[ip], atm->lat[ip]);
05154     for (int iq = 0; iq < ctl->nq; iq++) {
05155         fprintf(out, " ");
05156         if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05157             fprintf(out, ctl->qnt_format[iq], GSL_NAN);
05158         else
05159             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05160     }
05161     fprintf(out, "\n");
05162 }
05163
05164 /* Close file... */
05165 fclose(out);
05166 }

```

Here is the call graph for this function:



**5.21.3.69 write\_atm\_bin()** void write\_atm\_bin (
  
const char \* filename,
  
ctl\_t \* ctl,
  
atm\_t \* atm )

Write atmospheric data in binary format.

Definition at line 5170 of file libtrac.c.

```

05173     {
05174
05175     FILE *out;
05176
05177     /* Create file... */
05178     if (!(out = fopen(filename, "w")))
05179         ERRMSG("Cannot create file!");
05180
05181     /* Write version of binary data... */
05182     int version = 100;
05183     FWRITE(&version, int,
05184           1,

```

```

05185         out);
05186
05187     /* Write data... */
05188     FWRITE(&atm->np, int,
05189         1,
05190         out);
05191     FWRITE(atm->time, double,
05192         (size_t) atm->np,
05193         out);
05194     FWRITE(atm->p, double,
05195         (size_t) atm->np,
05196         out);
05197     FWRITE(atm->lon, double,
05198         (size_t) atm->np,
05199         out);
05200     FWRITE(atm->lat, double,
05201         (size_t) atm->np,
05202         out);
05203     for (int iq = 0; iq < ctl->nq; iq++)
05204         FWRITE(atm->q[iq], double,
05205             (size_t) atm->np,
05206             out);
05207
05208     /* Write final flag... */
05209     int final = 999;
05210     FWRITE(&final, int,
05211         1,
05212         out);
05213
05214     /* Close file... */
05215     fclose(out);
05216 }

```

**5.21.3.70 write\_atm\_clams()** void write\_atm\_clams (

```

    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write atmospheric data in CLaMS format.

Definition at line 5220 of file libtrac.c.

```

05223     {
05224
05225     /* Global Counter... */
05226     static size_t out_cnt = 0;
05227
05228     char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";
05229
05230     double r, r_start, r_stop;
05231
05232     int year, mon, day, hour, min, sec;
05233     int year_start, mon_start, day_start, hour_start, min_start, sec_start;
05234     int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
05235     int ncid, varid, tid, pid, cid, zid, dim_ids[2];
05236
05237     /* time, nparc */
05238     size_t start[2], count[2];
05239
05240     /* Determine start and stop times of calculation... */
05241     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05242     jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
05243         &min_start, &sec_start, &r_start);
05244     jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
05245         &min_stop, &sec_stop, &r_stop);
05246
05247     /* Set filename... */
05248     sprintf(filename_out,
05249         "/traj_fix_3d_%02d%02d%02d%02d%02d%02d%02d.nc",
05250         year_start % 100, mon_start, day_start, hour_start,
05251         year_stop % 100, mon_stop, day_stop, hour_stop);
05252     printf("Write traj file: %s\n", filename_out);
05253
05254     /* Define hyperslap for the traj_file... */
05255     start[0] = out_cnt;
05256     start[1] = 0;
05257     count[0] = 1;
05258     count[1] = (size_t) atm->np;
05259

```

```

05260  /* Create the file at the first timestep... */
05261  if (out_cnt == 0) {
05262
05263      /* Create file... */
05264      nc_create(filename_out, NC_CLOBBER, &ncid);
05265
05266      /* Define dimensions... */
05267      NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
05268      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05269      NC(nc_def_dim(ncid, "TMDT", 7, &cid));
05270      dim_ids[0] = tid;
05271      dim_ids[1] = pid;
05272
05273      /* Define variables and their attributes... */
05274      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05275                "seconds since 2000-01-01 00:00:00 UTC");
05276      NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
05277      NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
05278      NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
05279      NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
05280      for (int iq = 0; iq < ctl->nq; iq++)
05281          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05282                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05283
05284      /* Define global attributes... */
05285      NC_PUT_ATT("exp_VERTCOORD_name", "zeta");
05286      NC_PUT_ATT("model", "MPTRAC");
05287
05288      /* End definitions... */
05289      NC(nc_enddef(ncid));
05290      NC(nc_close(ncid));
05291  }
05292
05293  /* Increment global counter to change hyperslap... */
05294  out_cnt++;
05295
05296  /* Open file... */
05297  NC(nc_open(filename_out, NC_WRITE, &ncid));
05298
05299  /* Write data... */
05300  NC_PUT_DOUBLE("time", atm->time, 1);
05301  NC_PUT_DOUBLE("LAT", atm->lat, 1);
05302  NC_PUT_DOUBLE("LON", atm->lon, 1);
05303  NC_PUT_DOUBLE("PRESS", atm->p, 1);
05304  if (ctl->vert_coord_ap == 1) {
05305      NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
05306  } else if (ctl->qnt_zeta >= 0) {
05307      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
05308  }
05309  for (int iq = 0; iq < ctl->nq; iq++)
05310      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);
05311
05312  /* Close file... */
05313  NC(nc_close(ncid));
05314
05315  /* At the last time step create the init_fix_YYYYMMDDHH file... */
05316  if ((year == year_stop) && (mon == mon_stop)
05317      && (day == day_stop) && (hour == hour_stop)) {
05318
05319      /* Set filename... */
05320      char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
05321      sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
05322              year_stop % 100, mon_stop, day_stop, hour_stop);
05323      printf("Write init file: %s\n", filename_init);
05324
05325      /* Create file... */
05326      nc_create(filename_init, NC_CLOBBER, &ncid);
05327
05328      /* Define dimensions... */
05329      NC(nc_def_dim(ncid, "time", 1, &tid));
05330      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05331      dim_ids[0] = tid;
05332      dim_ids[1] = pid;
05333
05334      /* Define variables and their attributes... */
05335      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05336                "seconds since 2000-01-01 00:00:00 UTC");
05337      NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
05338      NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
05339      NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
05340      NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
05341      NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
05342      NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
05343      for (int iq = 0; iq < ctl->nq; iq++)
05344          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05345                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05346

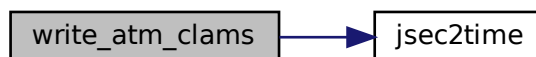
```

```

05347      /* Define global attributes... */
05348      NC_PUT_ATT("exp_VERTCOOR_name", "zeta");
05349      NC_PUT_ATT("model", "MPTRAC");
05350
05351      /* End definitions... */
05352      NC(nc_enddef(ncid));
05353
05354      /* Write data... */
05355      NC_PUT_DOUBLE("time", atm->time, 0);
05356      NC_PUT_DOUBLE("LAT", atm->lat, 0);
05357      NC_PUT_DOUBLE("LON", atm->lon, 0);
05358      NC_PUT_DOUBLE("PRESS", atm->p, 0);
05359      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
05360      for (int iq = 0; iq < ctl->nq; iq++)
05361          NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05362
05363      /* Close file... */
05364      NC(nc_close(ncid));
05365  }
05366 }

```

Here is the call graph for this function:



**5.21.3.71 write\_atm\_nc()** void write\_atm\_nc (

```

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

```

Write atmospheric data in netCDF format.

Definition at line 5370 of file libtrac.c.

```

05373      {
05374
05375      int ncid, obsid, varid;
05376
05377      size_t start[2], count[2];
05378
05379      /* Create file... */
05380      nc_create(filename, NC_CLOBBER, &ncid);
05381
05382      /* Define dimensions... */
05383      NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));
05384
05385      /* Define variables and their attributes... */
05386      NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
05387                "seconds since 2000-01-01 00:00:00 UTC");
05388      NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
05389      NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
05390      NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
05391      for (int iq = 0; iq < ctl->nq; iq++)
05392          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
05393                    ctl->qnt_longname[iq], ctl->qnt_unit[iq]);
05394
05395      /* Define global attributes... */
05396      NC_PUT_ATT("featureType", "point");
05397
05398      /* End definitions... */
05399      NC(nc_enddef(ncid));
05400
05401      /* Write data... */

```

```

05402 NC_PUT_DOUBLE("time", atm->time, 0);
05403 NC_PUT_DOUBLE("press", atm->p, 0);
05404 NC_PUT_DOUBLE("lon", atm->lon, 0);
05405 NC_PUT_DOUBLE("lat", atm->lat, 0);
05406 for (int iq = 0; iq < ctl->nq; iq++)
05407     NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05408
05409 /* Close file... */
05410 NC(nc_close(ncid));
05411 }

```

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

Write CSI data.

Definition at line 5415 of file libtrac.c.

```

05419 {
05420
05421     static FILE *out;
05422
05423     static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
05424         dlon, dlat, dz, x[NCSI], y[NCSI];
05425
05426     static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;
05427
05428     /* Set timer... */
05429     SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
05430
05431     /* Init... */
05432     if (t == ctl->t_start) {
05433
05434         /* Check quantity index for mass... */
05435         if (ctl->qnt_m < 0)
05436             ERRMSG("Need quantity mass!");
05437
05438         /* Allocate... */
05439         ALLOC(area, double,
05440             ctl->csi_ry);
05441         ALLOC(rt, double,
05442             NOBS);
05443         ALLOC(rz, double,
05444             NOBS);
05445         ALLOC(rlon, double,
05446             NOBS);
05447         ALLOC(rlat, double,
05448             NOBS);
05449         ALLOC(robs, double,
05450             NOBS);
05451
05452         /* Read observation data... */
05453         read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);
05454
05455         /* Create new file... */
05456         LOG(1, "Write CSI data: %s", filename);
05457         if (!(out = fopen(filename, "w")))
05458             ERRMSG("Cannot create file!");
05459
05460         /* Write header... */
05461         fprintf(out,
05462             "# $1 = time [s]\n"
05463             "# $2 = number of hits (cx)\n"
05464             "# $3 = number of misses (cy)\n"
05465             "# $4 = number of false alarms (cz)\n"
05466             "# $5 = number of observations (cx + cy)\n"
05467             "# $6 = number of forecasts (cx + cz)\n"
05468             "# $7 = bias (ratio of forecasts and observations) [%%]\n"
05469             "# $8 = probability of detection (POD) [%%]\n"
05470             "# $9 = false alarm rate (FAR) [%%]\n"
05471             "# $10 = critical success index (CSI) [%%]\n");
05472         fprintf(out,
05473             "# $11 = hits associated with random chance\n"
05474             "# $12 = equitable threat score (ETS) [%%]\n"
05475             "# $13 = Pearson linear correlation coefficient\n"

```

```

05476         "# $14 = Spearman rank-order correlation coefficient\n"
05477         "# $15 = column density mean error (F - O) [kg/m^2]\n"
05478         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
05479         "# $17 = column density mean absolute error [kg/m^2]\n"
05480         "# $18 = number of data points\n\n");
05481
05482     /* Set grid box size... */
05483     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
05484     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
05485     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
05486
05487     /* Set horizontal coordinates... */
05488     for (iy = 0; iy < ctl->csi_ny; iy++) {
05489         double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
05490         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
05491     }
05492 }
05493
05494 /* Set time interval... */
05495 double t0 = t - 0.5 * ctl->dt_mod;
05496 double t1 = t + 0.5 * ctl->dt_mod;
05497
05498 /* Allocate... */
05499 ALLOC(modmean, double,
05500       ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05501 ALLOC(obsmean, double,
05502       ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05503 ALLOC(obscount, int,
05504       ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05505
05506 /* Loop over observations... */
05507 for (int i = 0; i < nob; i++) {
05508
05509     /* Check time... */
05510     if (rt[i] < t0)
05511         continue;
05512     else if (rt[i] >= t1)
05513         break;
05514
05515     /* Check observation data... */
05516     if (!isfinite(robs[i]))
05517         continue;
05518
05519     /* Calculate indices... */
05520     ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
05521     iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
05522     iz = (int) ((rz[i] - ctl->csi_z0) / dz);
05523
05524     /* Check indices... */
05525     if (ix < 0 || ix >= ctl->csi_nx ||
05526         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05527         continue;
05528
05529     /* Get mean observation index... */
05530     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05531     obsmean[idx] += robs[i];
05532     obscount[idx]++;
05533 }
05534
05535 /* Analyze model data... */
05536 for (ip = 0; ip < atm->np; ip++) {
05537
05538     /* Check time... */
05539     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05540         continue;
05541
05542     /* Get indices... */
05543     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
05544     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
05545     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
05546
05547     /* Check indices... */
05548     if (ix < 0 || ix >= ctl->csi_nx ||
05549         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05550         continue;
05551
05552     /* Get total mass in grid cell... */
05553     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05554     modmean[idx] += atm->q[ctl->qnt_m][ip];
05555 }
05556
05557 /* Analyze all grid cells... */
05558 for (ix = 0; ix < ctl->csi_nx; ix++)
05559     for (iy = 0; iy < ctl->csi_ny; iy++)
05560         for (iz = 0; iz < ctl->csi_nz; iz++) {
05561
05562             /* Calculate mean observation index... */

```



```

05563     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05564     if (obscount[idx] > 0)
05565         obsmean[idx] /= obscount[idx];
05566
05567     /* Calculate column density... */
05568     if (modmean[idx] > 0)
05569         modmean[idx] /= (1e6 * area[iy]);
05570
05571     /* Calculate CSI... */
05572     if (obscount[idx] > 0) {
05573         ct++;
05574         if (obsmean[idx] >= ctl->csi_obsmin &&
05575             modmean[idx] >= ctl->csi_modmin)
05576             cx++;
05577         else if (obsmean[idx] >= ctl->csi_obsmin &&
05578                 modmean[idx] < ctl->csi_modmin)
05579             cy++;
05580         else if (obsmean[idx] < ctl->csi_obsmin &&
05581                 modmean[idx] >= ctl->csi_modmin)
05582             cz++;
05583     }
05584
05585     /* Save data for other verification statistics... */
05586     if (obscount[idx] > 0
05587         && (obsmean[idx] >= ctl->csi_obsmin
05588             || modmean[idx] >= ctl->csi_modmin)) {
05589         x[n] = modmean[idx];
05590         y[n] = obsmean[idx];
05591         if (++n > NCSI)
05592             ERRMSG("Too many data points to calculate statistics!");
05593     }
05594 }
05595
05596 /* Write output... */
05597 if (fmod(t, ctl->csi_dt_out) == 0) {
05598
05599     /* Calculate verification statistics
05600      (https://www.cawcr.gov.au/projects/verification/) ... */
05601     static double work[2 * NCSI];
05602     int n_obs = cx + cy;
05603     int n_for = cx + cz;
05604     double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
05605     double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
05606     double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
05607     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
05608     double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
05609     double ets = (cx + cy + cz - cx_rd > 0) ?
05610         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
05611     double rho_p =
05612         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
05613     double rho_s =
05614         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
05615     for (int i = 0; i < n; i++)
05616         work[i] = x[i] - y[i];
05617     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
05618     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
05619         0.0) : GSL_NAN;
05620     double absdev =
05621         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
05622
05623     /* Write... */
05624     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %d\n",
05625         t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
05626         rho_p, rho_s, mean, rmse, absdev, n);
05627
05628     /* Set counters to zero... */
05629     n = ct = cx = cy = cz = 0;
05630 }
05631
05632 /* Free... */
05633 free(modmean);
05634 free(obsmean);
05635 free(obscount);
05636
05637 /* Finalize... */
05638 if (t == ctl->t_stop) {
05639
05640     /* Close output file... */
05641     fclose(out);
05642
05643     /* Free... */
05644     free(area);
05645     free(rt);
05646     free(rz);
05647     free(rlon);
05648     free(rlat);
05649     free(robs);

```

```
05650     }
05651 }
```

Here is the call graph for this function:



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

Write ensemble data.

Definition at line 5655 of file libtrac.c.

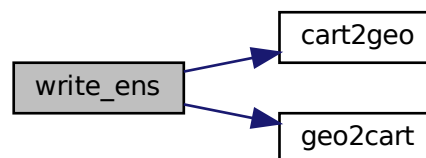
```
05659     {
05660
05661     static FILE *out;
05662
05663     static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
05664         x[3], zm[NENS];
05665
05666     static int n[NENS];
05667
05668     /* Set timer... */
05669     SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
05670
05671     /* Check quantities... */
05672     if (ctl->qnt_ens < 0)
05673         ERRMSG("Missing ensemble IDs!");
05674
05675     /* Set time interval... */
05676     double t0 = t - 0.5 * ctl->dt_mod;
05677     double t1 = t + 0.5 * ctl->dt_mod;
05678
05679     /* Init... */
05680     for (int i = 0; i < NENS; i++) {
05681         for (int iq = 0; iq < ctl->nq; iq++)
05682             qm[iq][i] = qs[iq][i] = 0;
05683         xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
05684         n[i] = 0;
05685     }
05686
05687     /* Loop over air parcels... */
05688     for (int ip = 0; ip < atm->np; ip++) {
05689
05690         /* Check time... */
05691         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05692             continue;
05693
05694         /* Check ensemble ID... */
05695         if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
05696             ERRMSG("Ensemble ID is out of range!");
05697
05698         /* Get means... */
05699         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
05700         for (int iq = 0; iq < ctl->nq; iq++) {
05701             qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
05702             qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
05703         }
05704         xm[ctl->qnt_ens][0] += x[0];
```

```

05705     xm[ctl->qnt_ens][1] += x[1];
05706     xm[ctl->qnt_ens][2] += x[2];
05707     zm[ctl->qnt_ens] += Z(atm->p[ip]);
05708     n[ctl->qnt_ens]++;
05709 }
05710
05711 /* Create file... */
05712 LOG(1, "Write ensemble data: %s", filename);
05713 if (!(out = fopen(filename, "w")))
05714     ERRMSG("Cannot create file!");
05715
05716 /* Write header... */
05717 fprintf(out,
05718         "# $1 = time [s]\n"
05719         "# $2 = altitude [km]\n"
05720         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05721 for (int iq = 0; iq < ctl->nq; iq++)
05722     fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
05723             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05724 for (int iq = 0; iq < ctl->nq; iq++)
05725     fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
05726             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05727 fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
05728
05729 /* Write data... */
05730 for (int i = 0; i < NENS; i++)
05731     if (n[i] > 0) {
05732         cart2geo(xm[i], &dummy, &lon, &lat);
05733         fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
05734         for (int iq = 0; iq < ctl->nq; iq++) {
05735             fprintf(out, " ");
05736             fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
05737         }
05738         for (int iq = 0; iq < ctl->nq; iq++) {
05739             fprintf(out, " ");
05740             double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
05741             fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
05742         }
05743         fprintf(out, " %d\n", n[i]);
05744     }
05745
05746 /* Close file... */
05747 fclose(out);
05748 }

```

Here is the call graph for this function:



**5.21.3.74 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 5752 of file libtrac.c.

```

05758     {
05759
05760     double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;
05761
05762     int *ixs, *iys, *izs, *np;
05763
05764     /* Set timer... */
05765     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
05766
05767     /* Write info... */
05768     LOG(1, "Write grid data: %s", filename);
05769
05770     /* Allocate... */
05771     ALLOC(cd, double,
05772           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05773     ALLOC(mass, double,
05774           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05775     ALLOC(vmr_expl, double,
05776           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05777     ALLOC(vmr_impl, double,
05778           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05779     ALLOC(z, double,
05780           ctl->grid_nz);
05781     ALLOC(lon, double,
05782           ctl->grid_nx);
05783     ALLOC(lat, double,
05784           ctl->grid_ny);
05785     ALLOC(area, double,
05786           ctl->grid_ny);
05787     ALLOC(press, double,
05788           ctl->grid_nz);
05789     ALLOC(np, int,
05790           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05791     ALLOC(ixs, int,
05792           atm->np);
05793     ALLOC(iys, int,
05794           atm->np);
05795     ALLOC(izs, int,
05796           atm->np);
05797
05798     /* Set grid box size... */
05799     double dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
05800     double dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
05801     double dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
05802
05803     /* Set vertical coordinates... */
05804     #pragma omp parallel for default(shared)
05805     for (int iz = 0; iz < ctl->grid_nz; iz++) {
05806         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
05807         press[iz] = P(z[iz]);
05808     }
05809
05810     /* Set horizontal coordinates... */
05811     for (int ix = 0; ix < ctl->grid_nx; ix++)
05812         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
05813     #pragma omp parallel for default(shared)
05814     for (int iy = 0; iy < ctl->grid_ny; iy++) {
05815         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
05816         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05817             * cos(lat[iy] * M_PI / 180.);
05818     }
05819
05820     /* Set time interval for output... */
05821     double t0 = t - 0.5 * ctl->dt_mod;
05822     double t1 = t + 0.5 * ctl->dt_mod;
05823
05824     /* Get grid box indices... */
05825     #pragma omp parallel for default(shared)
05826     for (int ip = 0; ip < atm->np; ip++) {
05827         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
05828         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
05829         izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05830         if (atm->time[ip] < t0 || atm->time[ip] > t1
05831             || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05832             || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05833             || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05834             izs[ip] = -1;
05835     }
05836
05837     /* Average data... */
05838     for (int ip = 0; ip < atm->np; ip++)
05839         if (izs[ip] >= 0) {
05840             int idx =
05841                 ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
05842             np[idx]++;
05843             if (ctl->qnt_m >= 0)

```

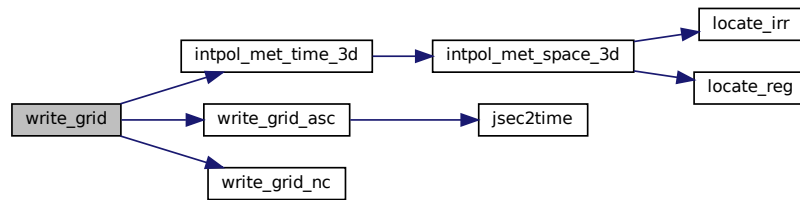
```

05844     mass[idx] += atm->q[ctl->qnt_m][ip];
05845     if (ctl->qnt_vmr >= 0)
05846         vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
05847 }
05848
05849 /* Get implicit vmr per particle... */
05850 if (ctl->qnt_vmrimpl >= 0)
05851     for (int ip = 0; ip < atm->np; ip++)
05852         if (izs[ip] >= 0) {
05853             double temp;
05854             INTPOL_INIT;
05855             intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
05856                             lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
05857             atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass
05858                 *
05859                 mass[ARRAY_3D
05860                     (ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz)]
05861                 / (RHO(press[izs[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
05862         }
05863
05864 /* Calculate column density and vmr... */
05865 #pragma omp parallel for default(shared)
05866 for (int ix = 0; ix < ctl->grid_nx; ix++)
05867     for (int iy = 0; iy < ctl->grid_ny; iy++)
05868         for (int iz = 0; iz < ctl->grid_nz; iz++) {
05869
05870             /* Get grid index... */
05871             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
05872
05873             /* Calculate column density... */
05874             cd[idx] = GSL_NAN;
05875             if (ctl->qnt_m >= 0)
05876                 cd[idx] = mass[idx] / (1e6 * area[iy]);
05877
05878             /* Calculate volume mixing ratio (implicit)... */
05879             vmr_impl[idx] = GSL_NAN;
05880             if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05881                 vmr_impl[idx] = 0;
05882                 if (mass[idx] > 0) {
05883
05884                     /* Get temperature... */
05885                     double temp;
05886                     INTPOL_INIT;
05887                     intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05888                                     lon[ix], lat[iy], &temp, ci, cw, 1);
05889
05890                     /* Calculate volume mixing ratio... */
05891                     vmr_impl[idx] = MA / ctl->molmass * mass[idx]
05892                                     / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
05893                 }
05894             }
05895
05896             /* Calculate volume mixing ratio (explicit)... */
05897             if (ctl->qnt_vmr >= 0 && np[idx] > 0)
05898                 vmr_expl[idx] /= np[idx];
05899             else
05900                 vmr_expl[idx] = GSL_NAN;
05901         }
05902
05903 /* Write ASCII data... */
05904 if (ctl->grid_type == 0)
05905     write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
05906                  t, z, lon, lat, area, dz, np);
05907
05908 /* Write netCDF data... */
05909 else if (ctl->grid_type == 1)
05910     write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
05911                  t, z, lon, lat, area, dz, np);
05912
05913 /* Error message... */
05914 else
05915     ERRMSG("Grid data format GRID_TYPE unknown!");
05916
05917 /* Free... */
05918 free(cd);
05919 free(mass);
05920 free(vmr_expl);
05921 free(vmr_impl);
05922 free(z);
05923 free(lon);
05924 free(lat);
05925 free(area);
05926 free(press);
05927 free(np);
05928 free(ixs);
05929 free(iys);
05930 free(izs);

```

```
05931 }
```

Here is the call graph for this function:



**5.21.3.75 write\_grid\_asc()** void write\_grid\_asc (

```

    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )

```

Write gridded data in ASCII format.

Definition at line 5935 of file libtrac.c.

```

05947     {
05948
05949     FILE *in, *out;
05950
05951     char line[LEN];
05952
05953     /* Check if gnuplot output is requested... */
05954     if (ctl->grid_gpfile[0] != '-') {
05955
05956         /* Create gnuplot pipe... */
05957         if (!(out = popen("gnuplot", "w")))
05958             ERRMSG("Cannot create pipe to gnuplot!");
05959
05960         /* Set plot filename... */
05961         fprintf(out, "set out \"%s.png\"\n", filename);
05962
05963         /* Set time string... */
05964         double r;
05965         int year, mon, day, hour, min, sec;
05966         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05967         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05968             year, mon, day, hour, min);
05969
05970         /* Dump gnuplot file to pipe... */
05971         if (!(in = fopen(ctl->grid_gpfile, "r")))
05972             ERRMSG("Cannot open file!");
05973         while (fgets(line, LEN, in))
05974             fprintf(out, "%s", line);
05975         fclose(in);
05976     }
05977
05978     else {

```

```

05979
05980     /* Create file... */
05981     if (!out = fopen(filename, "w"))
05982         ERRMSG("Cannot create file!");
05983 }
05984
05985 /* Write header... */
05986 fprintf(out,
05987     "# $1 = time [s]\n"
05988     "# $2 = altitude [km]\n"
05989     "# $3 = longitude [deg]\n"
05990     "# $4 = latitude [deg]\n"
05991     "# $5 = surface area [km^2]\n"
05992     "# $6 = layer depth [km]\n"
05993     "# $7 = number of particles [l]\n"
05994     "# $8 = column density (implicit) [kg/m^2]\n"
05995     "# $9 = volume mixing ratio (implicit) [ppv]\n"
05996     "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
05997
05998 /* Write data... */
05999 for (int ix = 0; ix < ctl->grid_nx; ix++) {
06000     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
06001         fprintf(out, "\n");
06002     for (int iy = 0; iy < ctl->grid_ny; iy++) {
06003         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
06004             fprintf(out, "\n");
06005         for (int iz = 0; iz < ctl->grid_nz; iz++) {
06006             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
06007             if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
06008                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
06009                     t, z[iz], lon[ix], lat[iy], area[iy], dz,
06010                     np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
06011         }
06012     }
06013 }
06014
06015 /* Close file... */
06016 fclose(out);
06017 }

```

Here is the call graph for this function:



### 5.21.3.76 write\_grid\_nc() void write\_grid\_nc (

```

const char * filename,
ctl_t * ctl,
double * cd,
double * vmr_expl,
double * vmr_impl,
double t,
double * z,
double * lon,
double * lat,
double * area,
double dz,
int * np )

```

Write gridded data in netCDF format.

Definition at line 6021 of file `libtrac.c`.

```

06033     {
06034
06035     int ncid, dimid[10], varid;
06036
06037     size_t start[2], count[2];
06038
06039     /* Create file... */
06040     nc_create(filename, NC_CLOBBER, &ncid);
06041
06042     /* Define dimensions... */
06043     NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
06044     NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[1]));
06045     NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
06046     NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[3]));
06047     NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));
06048
06049     /* Define variables and their attributes... */
06050     NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
06051               "seconds since 2000-01-01 00:00:00 UTC");
06052     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[1], "longitude", "degrees_east");
06053     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
06054     NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[3], "altitude", "km");
06055     NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
06056     NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[4], "layer depth", "km");
06057     NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
06058     NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
06059               "volume mixing ratio (implicit)", "ppv");
06060     NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
06061               "volume mixing ratio (explicit)", "ppv");
06062     NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");
06063
06064     /* End definitions... */
06065     NC(nc_enddef(ncid));
06066
06067     /* Write data... */
06068     NC_PUT_DOUBLE("time", &t, 0);
06069     NC_PUT_DOUBLE("lon", lon, 0);
06070     NC_PUT_DOUBLE("lat", lat, 0);
06071     NC_PUT_DOUBLE("z", z, 0);
06072     NC_PUT_DOUBLE("area", area, 0);
06073     NC_PUT_DOUBLE("dz", &dz, 0);
06074     NC_PUT_DOUBLE("cd", cd, 0);
06075     NC_PUT_DOUBLE("vmr_impl", vmr_impl, 0);
06076     NC_PUT_DOUBLE("vmr_expl", vmr_expl, 0);
06077     NC_PUT_INT("np", np, 0);
06078
06079     /* Close file... */
06080     NC(nc_close(ncid));
06081 }

```

**5.21.3.77 write\_met()** int write\_met (  
     char \* filename,  
     ctl\_t \* ctl,  
     met\_t \* met )

Read meteo data file.

Definition at line 6085 of file `libtrac.c`.

```

06088     {
06089
06090     /* Set timer... */
06091     SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);
06092
06093     /* Write info... */
06094     LOG(1, "Write meteo data: %s", filename);
06095
06096     /* Check compression flags... */
06097     #ifndef ZFP
06098         if (ctl->met_type == 3)
06099             ERRMSG("zfp compression not supported!");
06100     #endif
06101     #ifndef ZSTD
06102         if (ctl->met_type == 4)
06103             ERRMSG("zstd compression not supported!");
06104     #endif
06105
06106     /* Write binary... */

```

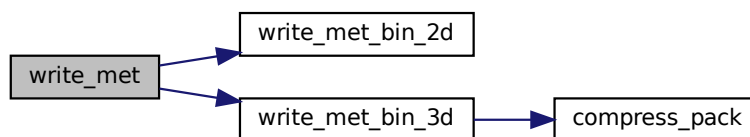


```

06107     if (ctl->met_type >= 1 && ctl->met_type <= 4) {
06108
06109         /* Create file... */
06110         FILE *out;
06111         if (!(out = fopen(filename, "w")))
06112             ERRMSG("Cannot create file!");
06113
06114         /* Write type of binary data... */
06115         FWRITE(&ctl->met_type, int,
06116             1,
06117             out);
06118
06119         /* Write version of binary data... */
06120         int version = 100;
06121         FWRITE(&version, int,
06122             1,
06123             out);
06124
06125         /* Write grid data... */
06126         FWRITE(&met->time, double,
06127             1,
06128             out);
06129         FWRITE(&met->nx, int,
06130             1,
06131             out);
06132         FWRITE(&met->ny, int,
06133             1,
06134             out);
06135         FWRITE(&met->np, int,
06136             1,
06137             out);
06138         FWRITE(met->lon, double,
06139             (size_t) met->nx,
06140             out);
06141         FWRITE(met->lat, double,
06142             (size_t) met->ny,
06143             out);
06144         FWRITE(met->p, double,
06145             (size_t) met->np,
06146             out);
06147
06148         /* Write surface data... */
06149         write_met_bin_2d(out, met, met->ps, "PS");
06150         write_met_bin_2d(out, met, met->ts, "TS");
06151         write_met_bin_2d(out, met, met->zs, "ZS");
06152         write_met_bin_2d(out, met, met->us, "US");
06153         write_met_bin_2d(out, met, met->vs, "VS");
06154         write_met_bin_2d(out, met, met->pbl, "PBL");
06155         write_met_bin_2d(out, met, met->pt, "PT");
06156         write_met_bin_2d(out, met, met->tt, "TT");
06157         write_met_bin_2d(out, met, met->zt, "ZT");
06158         write_met_bin_2d(out, met, met->h2ot, "H2OT");
06159         write_met_bin_2d(out, met, met->pct, "PCT");
06160         write_met_bin_2d(out, met, met->pcb, "PCB");
06161         write_met_bin_2d(out, met, met->cl, "CL");
06162         write_met_bin_2d(out, met, met->plcl, "PLCL");
06163         write_met_bin_2d(out, met, met->plfc, "PLFC");
06164         write_met_bin_2d(out, met, met->pel, "PEL");
06165         write_met_bin_2d(out, met, met->cape, "CAPE");
06166         write_met_bin_2d(out, met, met->cin, "CIN");
06167
06168         /* Write level data... */
06169         write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
06170         write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
06171         write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
06172         write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
06173         write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
06174         write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
06175         write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
06176         write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
06177         write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
06178         write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);
06179
06180         /* Write final flag... */
06181         int final = 999;
06182         FWRITE(&final, int,
06183             1,
06184             out);
06185
06186         /* Close file... */
06187         fclose(out);
06188     }
06189
06190     return 0;
06191 }

```

Here is the call graph for this function:



**5.21.3.78 write\_met\_bin\_2d()** void write\_met\_bin\_2d (  
 FILE \* out,  
 met\_t \* met,  
 float var[EX][EY],  
 char \* varname )

Write 2-D meteo variable.

Definition at line 6195 of file libtrac.c.

```

06199     {
06200
06201     float *help;
06202
06203     /* Allocate... */
06204     ALLOC(help, float,
06205           EX * EY);
06206
06207     /* Copy data... */
06208     for (int ix = 0; ix < met->nx; ix++)
06209         for (int iy = 0; iy < met->ny; iy++)
06210             help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];
06211
06212     /* Write uncompressed data... */
06213     LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
06214     FWRITE(help, float,
06215            (size_t) (met->nx * met->ny),
06216            out);
06217
06218     /* Free... */
06219     free(help);
06220 }
  
```

**5.21.3.79 write\_met\_bin\_3d()** void write\_met\_bin\_3d (  
 FILE \* out,  
 ctl\_t \* ctl,  
 met\_t \* met,  
 float var[EX][EY][EP],  
 char \* varname,  
 int precision,  
 double tolerance )

Write 3-D meteo variable.

Definition at line 6224 of file libtrac.c.

```

06231     {
  
```

```

06232
06233     float *help;
06234
06235     /* Allocate... */
06236     ALLOC(help, float,
06237           EX * EY * EP);
06238
06239     /* Copy data... */
06240     #pragma omp parallel for default(shared) collapse(2)
06241     for (int ix = 0; ix < met->nx; ix++)
06242         for (int iy = 0; iy < met->ny; iy++)
06243             for (int ip = 0; ip < met->np; ip++)
06244                 help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];
06245
06246     /* Write uncompressed data... */
06247     if (ctl->met_type == 1) {
06248         LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
06249         FWRITE(help, float,
06250               (size_t) (met->nx * met->ny * met->np),
06251               out);
06252     }
06253
06254     /* Write packed data... */
06255     else if (ctl->met_type == 2)
06256         compress_pack(varname, help, (size_t) (met->ny * met->nx),
06257                       (size_t) met->np, 0, out);
06258
06259     /* Write zfp data... */
06260     #ifndef ZFP
06261     else if (ctl->met_type == 3)
06262         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
06263                     tolerance, 0, out);
06264     #endif
06265
06266     /* Write zstd data... */
06267     #ifndef ZSTD
06268     else if (ctl->met_type == 4)
06269         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
06270                      out);
06271     #endif
06272
06273     /* Unknown method... */
06274     else {
06275         ERRMSG("MET_TYPE not supported!");
06276         LOG(3, "%d %g", precision, tolerance);
06277     }
06278
06279     /* Free... */
06280     free(help);
06281 }

```

Here is the call graph for this function:



**5.21.3.80 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 6285 of file libtrac.c.

```

6291     {
6292
6293     static FILE *out;
6294
6295     static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
6296         dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;
6297
6298     static int nob, *obscount, ip, okay;
6299
6300     /* Set timer... */
6301     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
6302
6303     /* Init... */
6304     if (t == ctl->t_start) {
6305
6306         /* Check quantity index for mass... */
6307         if (ctl->qnt_m < 0)
6308             ERRMSG("Need quantity mass!");
6309
6310         /* Check molar mass... */
6311         if (ctl->molmass <= 0)
6312             ERRMSG("Specify molar mass!");
6313
6314         /* Allocate... */
6315         ALLOC(lon, double,
6316             ctl->prof_nx);
6317         ALLOC(lat, double,
6318             ctl->prof_ny);
6319         ALLOC(area, double,
6320             ctl->prof_ny);
6321         ALLOC(z, double,
6322             ctl->prof_nz);
6323         ALLOC(press, double,
6324             ctl->prof_nz);
6325         ALLOC(rt, double,
6326             NOBS);
6327         ALLOC(rz, double,
6328             NOBS);
6329         ALLOC(rlon, double,
6330             NOBS);
6331         ALLOC(rlat, double,
6332             NOBS);
6333         ALLOC(robs, double,
6334             NOBS);
6335
6336         /* Read observation data... */
6337         read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nob);
6338
6339         /* Create new output file... */
6340         LOG(1, "Write profile data: %s", filename);
6341         if (!(out = fopen(filename, "w")))
6342             ERRMSG("Cannot create file!");
6343
6344         /* Write header... */
6345         fprintf(out,
6346             "# $1 = time [s]\n"
6347             "# $2 = altitude [km]\n"
6348             "# $3 = longitude [deg]\n"
6349             "# $4 = latitude [deg]\n"
6350             "# $5 = pressure [hPa]\n"
6351             "# $6 = temperature [K]\n"
6352             "# $7 = volume mixing ratio [ppv]\n"
6353             "# $8 = H2O volume mixing ratio [ppv]\n"
6354             "# $9 = O3 volume mixing ratio [ppv]\n"
6355             "# $10 = observed BT index [K]\n"
6356             "# $11 = number of observations\n");
6357
6358         /* Set grid box size... */
6359         dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
6360         dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
6361         dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
6362
6363         /* Set vertical coordinates... */
6364         for (int iz = 0; iz < ctl->prof_nz; iz++) {
6365             z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
6366             press[iz] = P(z[iz]);
6367         }
6368
6369         /* Set horizontal coordinates... */
6370         for (int ix = 0; ix < ctl->prof_nx; ix++)
6371             lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
6372         for (int iy = 0; iy < ctl->prof_ny; iy++) {
6373             lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);

```

```

06374         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
06375         * cos(lat[iy] * M_PI / 180.);
06376     }
06377 }
06378
06379 /* Set time interval... */
06380 double t0 = t - 0.5 * ctl->dt_mod;
06381 double t1 = t + 0.5 * ctl->dt_mod;
06382
06383 /* Allocate... */
06384 ALLOC(mass, double,
06385       ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
06386 ALLOC(obsmean, double,
06387       ctl->prof_nx * ctl->prof_ny);
06388 ALLOC(obscount, int,
06389       ctl->prof_nx * ctl->prof_ny);
06390
06391 /* Loop over observations... */
06392 for (int i = 0; i < nob; i++) {
06393
06394     /* Check time... */
06395     if (rt[i] < t0)
06396         continue;
06397     else if (rt[i] >= t1)
06398         break;
06399
06400     /* Check observation data... */
06401     if (!isfinite(robs[i]))
06402         continue;
06403
06404     /* Calculate indices... */
06405     int ix = (int) ((rlon[i] - ctl->prof_lon0) / dlon);
06406     int iy = (int) ((rlat[i] - ctl->prof_lat0) / dlat);
06407
06408     /* Check indices... */
06409     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
06410         continue;
06411
06412     /* Get mean observation index... */
06413     int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
06414     obsmean[idx] += robs[i];
06415     obscount[idx]++;
06416 }
06417
06418 /* Analyze model data... */
06419 for (ip = 0; ip < atm->np; ip++) {
06420
06421     /* Check time... */
06422     if (atm->time[ip] < t0 || atm->time[ip] > t1)
06423         continue;
06424
06425     /* Get indices... */
06426     int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
06427     int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
06428     int iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
06429
06430     /* Check indices... */
06431     if (ix < 0 || ix >= ctl->prof_nx ||
06432         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
06433         continue;
06434
06435     /* Get total mass in grid cell... */
06436     int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06437     mass[idx] += atm->q[ctl->qnt_m][ip];
06438 }
06439
06440 /* Extract profiles... */
06441 for (int ix = 0; ix < ctl->prof_nx; ix++)
06442     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06443         int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
06444         if (obscount[idx2] > 0) {
06445
06446             /* Check profile... */
06447             okay = 0;
06448             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06449                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06450                 if (mass[idx3] > 0) {
06451                     okay = 1;
06452                     break;
06453                 }
06454             }
06455             if (!okay)
06456                 continue;
06457
06458             /* Write output... */
06459             fprintf(out, "\n");
06460

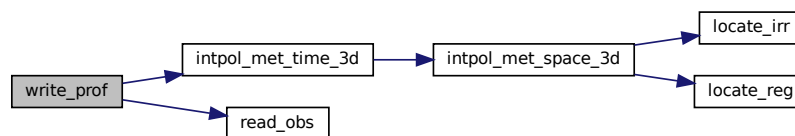
```

```

06461      /* Loop over altitudes... */
06462      for (int iz = 0; iz < ctl->prof_nz; iz++) {
06463
06464          /* Get temperature, water vapor, and ozone... */
06465          INTPOL_INIT;
06466          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
06467                          lon[ix], lat[iy], &temp, ci, cw, 1);
06468          intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
06469                          lon[ix], lat[iy], &h2o, ci, cw, 0);
06470          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
06471                          lon[ix], lat[iy], &o3, ci, cw, 0);
06472
06473          /* Calculate volume mixing ratio... */
06474          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06475          vmr = MA / ctl->molmass * mass[idx3]
06476              / (RHO(press[iz], temp) * area[iy] * dz * 1e9);
06477
06478          /* Write output... */
06479          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
06480                  t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
06481                  obsmean[idx2] / obscount[idx2], obscount[idx2]);
06482      }
06483  }
06484  }
06485
06486  /* Free... */
06487  free(mass);
06488  free(obsmean);
06489  free(obscount);
06490
06491  /* Finalize... */
06492  if (t == ctl->t_stop) {
06493
06494      /* Close output file... */
06495      fclose(out);
06496
06497      /* Free... */
06498      free(lon);
06499      free(lat);
06500      free(area);
06501      free(z);
06502      free(press);
06503      free(rt);
06504      free(rz);
06505      free(rlon);
06506      free(rlat);
06507      free(robs);
06508  }
06509  }

```

Here is the call graph for this function:



**5.21.3.81 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 6513 of file libtrac.c.

```

06519     {
06520
06521     static FILE *out;
06522
06523     static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;
06524
06525     static int nobs;
06526
06527     /* Set timer... */
06528     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
06529
06530     /* Init... */
06531     if (t == ctl->t_start) {
06532
06533         /* Allocate... */
06534         ALLOC(rt, double,
06535              NOBS);
06536         ALLOC(rz, double,
06537              NOBS);
06538         ALLOC(rlon, double,
06539              NOBS);
06540         ALLOC(rlat, double,
06541              NOBS);
06542         ALLOC(robs, double,
06543              NOBS);
06544
06545         /* Read observation data... */
06546         read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06547
06548         /* Create output file... */
06549         LOG(l, "Write sample data: %s", filename);
06550         if (!(out = fopen(filename, "w")))
06551             ERRMSG("Cannot create file!");
06552
06553         /* Write header... */
06554         fprintf(out,
06555              "# $1 = time [s]\n"
06556              "# $2 = altitude [km]\n"
06557              "# $3 = longitude [deg]\n"
06558              "# $4 = latitude [deg]\n"
06559              "# $5 = surface area [km^2]\n"
06560              "# $6 = layer depth [km]\n"
06561              "# $7 = number of particles [1]\n"
06562              "# $8 = column density [kg/m^2]\n"
06563              "# $9 = volume mixing ratio [ppv]\n"
06564              "# $10 = observed BT index [K]\n\n");
06565
06566         /* Set latitude range, squared radius, and area... */
06567         dlat = DY2DEG(ctl->sample_dx);
06568         rmax2 = SQR(ctl->sample_dx);
06569         area = M_PI * rmax2;
06570     }
06571
06572     /* Set time interval for output... */
06573     double t0 = t - 0.5 * ctl->dt_mod;
06574     double t1 = t + 0.5 * ctl->dt_mod;
06575
06576     /* Loop over observations... */
06577     for (int i = 0; i < nobs; i++) {
06578
06579         /* Check time... */
06580         if (rt[i] < t0)
06581             continue;
06582         else if (rt[i] >= t1)
06583             break;
06584
06585         /* Calculate Cartesian coordinates... */
06586         double x0[3];
06587         geo2cart(0, rlon[i], rlat[i], x0);
06588
06589         /* Set pressure range... */
06590         double rp = P(rz[i]);
06591         double ptop = P(rz[i] + ctl->sample_dz);
06592         double pbot = P(rz[i] - ctl->sample_dz);
06593
06594         /* Init... */
06595         double mass = 0;
06596         int np = 0;
06597
06598         /* Loop over air parcels... */
06599 #pragma omp parallel for default(shared) reduction(+:mass,np)
06600         for (int ip = 0; ip < atm->np; ip++) {
06601

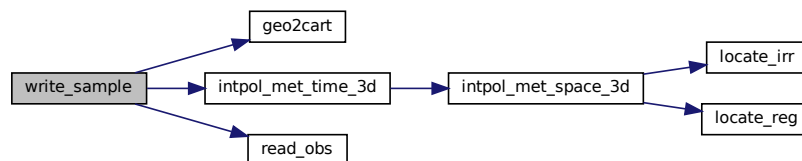
```

```

06602      /* Check time... */
06603      if (atm->time[ip] < t0 || atm->time[ip] > t1)
06604          continue;
06605
06606      /* Check latitude... */
06607      if (fabs(rlat[i] - atm->lat[ip]) > dlat)
06608          continue;
06609
06610      /* Check horizontal distance... */
06611      double x1[3];
06612      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06613      if (DIST2(x0, x1) > rmax2)
06614          continue;
06615
06616      /* Check pressure... */
06617      if (ctl->sample_dz > 0)
06618          if (atm->p[ip] > pbot || atm->p[ip] < ptop)
06619              continue;
06620
06621      /* Add mass... */
06622      if (ctl->qnt_m >= 0)
06623          mass += atm->q[ctl->qnt_m][ip];
06624      np++;
06625  }
06626
06627      /* Calculate column density... */
06628      double cd = mass / (1e6 * area);
06629
06630      /* Calculate volume mixing ratio... */
06631      double vmr = 0;
06632      if (ctl->molmass > 0 && ctl->sample_dz > 0) {
06633          if (mass > 0) {
06634
06635              /* Get temperature... */
06636              double temp;
06637              INTPOL_INIT;
06638              intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
06639                               rlon[i], rlat[i], &temp, ci, cw, 1);
06640
06641              /* Calculate volume mixing ratio... */
06642              vmr = MA / ctl->molmass * mass
06643                  / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
06644          }
06645      } else
06646          vmr = GSL_NAN;
06647
06648      /* Write output... */
06649      fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
06650              rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
06651  }
06652
06653      /* Finalize..... */
06654      if (t == ctl->t_stop) {
06655
06656          /* Close output file... */
06657          fclose(out);
06658
06659          /* Free... */
06660          free(rt);
06661          free(rz);
06662          free(rlon);
06663          free(rlat);
06664          free(robs);
06665      }
06666  }

```

Here is the call graph for this function:





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

Write station data.

Definition at line 6670 of file libtrac.c.

```

06674     {
06675
06676     static FILE *out;
06677
06678     static double rmax2, x0[3], x1[3];
06679
06680     /* Set timer... */
06681     SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
06682
06683     /* Init... */
06684     if (t == ctl->t_start) {
06685
06686         /* Write info... */
06687         LOG(1, "Write station data: %s", filename);
06688
06689         /* Create new file... */
06690         if (!(out = fopen(filename, "w")))
06691             ERRMSG("Cannot create file!");
06692
06693         /* Write header... */
06694         fprintf(out,
06695             "# $1 = time [s]\n"
06696             "# $2 = altitude [km]\n"
06697             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
06698         for (int iq = 0; iq < ctl->nq; iq++)
06699             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
06700                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
06701         fprintf(out, "\n");
06702
06703         /* Set geolocation and search radius... */
06704         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
06705         rmax2 = SQR(ctl->stat_r);
06706     }
06707
06708     /* Set time interval for output... */
06709     double t0 = t - 0.5 * ctl->dt_mod;
06710     double t1 = t + 0.5 * ctl->dt_mod;
06711
06712     /* Loop over air parcels... */
06713     for (int ip = 0; ip < atm->np; ip++) {
06714
06715         /* Check time... */
06716         if (atm->time[ip] < t0 || atm->time[ip] > t1)
06717             continue;
06718
06719         /* Check time range for station output... */
06720         if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
06721             continue;
06722
06723         /* Check station flag... */
06724         if (ctl->qnt_stat >= 0)
06725             if (atm->q[ctl->qnt_stat][ip])
06726                 continue;
06727
06728         /* Get Cartesian coordinates... */
06729         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06730
06731         /* Check horizontal distance... */
06732         if (DIST2(x0, x1) > rmax2)
06733             continue;
06734
06735         /* Set station flag... */
06736         if (ctl->qnt_stat >= 0)
06737             atm->q[ctl->qnt_stat][ip] = 1;
06738
06739         /* Write data... */
06740         fprintf(out, "%.2f %g %g %g",
06741             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
06742         for (int iq = 0; iq < ctl->nq; iq++) {
06743             fprintf(out, " ");
06744             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
06745         }
06746         fprintf(out, "\n");
06747     }

```

```

06748
06749  /* Close file... */
06750  if (t == ctl->t_stop)
06751      fclose(out);
06752  }

```

Here is the call graph for this function:



## 5.22 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) 2013–2022 Forschungszentrum Juelich GmbH
00018  */
00019
00037  #ifndef LIBTRAC_H
00038  #define LIBTRAC_H
00039
00040  /* -----
00041   Includes...
00042   ----- */
00043
00044  #include <ctype.h>
00045  #include <gsl/gsl_fft_complex.h>
00046  #include <gsl/gsl_math.h>
00047  #include <gsl/gsl_randist.h>
00048  #include <gsl/gsl_rng.h>
00049  #include <gsl/gsl_spline.h>
00050  #include <gsl/gsl_statistics.h>
00051  #include <math.h>
00052  #include <netcdf.h>
00053  #include <omp.h>
00054  #include <stdio.h>
00055  #include <stdlib.h>
00056  #include <string.h>
00057  #include <time.h>
00058  #include <sys/time.h>
00059
00060  #ifdef MPI
00061  #include "mpi.h"
00062  #endif
00063
00064  #ifdef _OPENACC
00065  #include "openacc.h"
00066  #include "curand.h"
00067  #endif
00068
00069  #ifdef ZFP
00070  #include "zfp.h"
00071  #endif
00072
00073  #ifdef ZSTD
00074  #include "zstd.h"
00075  #endif

```

```
00076
00077 /* -----
00078     Constants...
00079     ----- */
00080
00082 #ifndef CPD
00083 #define CPD 1003.5
00084 #endif
00085
00087 #ifndef EPS
00088 #define EPS (MH2O / MA)
00089 #endif
00090
00092 #ifndef G0
00093 #define G0 9.80665
00094 #endif
00095
00097 #ifndef H0
00098 #define H0 7.0
00099 #endif
00100
00102 #ifndef LV
00103 #define LV 2501000.
00104 #endif
00105
00107 #ifndef KB
00108 #define KB 1.3806504e-23
00109 #endif
00110
00112 #ifndef MA
00113 #define MA 28.9644
00114 #endif
00115
00117 #ifndef MH2O
00118 #define MH2O 18.01528
00119 #endif
00120
00122 #ifndef MO3
00123 #define MO3 48.00
00124 #endif
00125
00127 #ifndef P0
00128 #define P0 1013.25
00129 #endif
00130
00132 #ifndef RA
00133 #define RA (1e3 * RI / MA)
00134 #endif
00135
00137 #ifndef RE
00138 #define RE 6367.421
00139 #endif
00140
00142 #ifndef RI
00143 #define RI 8.3144598
00144 #endif
00145
00147 #ifndef T0
00148 #define T0 273.15
00149 #endif
00150
00151 /* -----
00152     Dimensions...
00153     ----- */
00154
00156 #ifndef LEN
00157 #define LEN 5000
00158 #endif
00159
00161 #ifndef NP
00162 #define NP 10000000
00163 #endif
00164
00166 #ifndef NQ
00167 #define NQ 15
00168 #endif
00169
00171 #ifndef NCSI
00172 #define NCSI 1000000
00173 #endif
00174
00176 #ifndef EP
00177 #define EP 140
00178 #endif
00179
00181 #ifndef EX
00182 #define EX 1201
```

```

00183 #endif
00184
00186 #ifndef EY
00187 #define EY 601
00188 #endif
00189
00191 #ifndef NENS
00192 #define NENS 2000
00193 #endif
00194
00196 #ifndef NOBS
00197 #define NOBS 10000000
00198 #endif
00199
00201 #ifndef NTHREADS
00202 #define NTHREADS 512
00203 #endif
00204
00206 #ifndef CY
00207 #define CY 250
00208 #endif
00209
00211 #ifndef CP
00212 #define CP 60
00213 #endif
00214
00216 #ifndef CT
00217 #define CT 12
00218 #endif
00219
00220 /* -----
00221     Macros...
00222     ----- */
00223
00225 #ifdef _OPENACC
00226 #define ALLOC(ptr, type, n) \
00227     if(acc_get_num_devices(acc_device_nvidia) <= 0) \
00228         ERRMSG("Not running on a GPU device!"); \
00229     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00230         ERRMSG("Out of memory!");
00231 #else
00232 #define ALLOC(ptr, type, n) \
00233     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00234         ERRMSG("Out of memory!");
00235 #endif
00236
00238 #define ARRAY_2D(ix, iy, ny) \
00239     ((ix) * (ny) + (iy))
00240
00242 #define ARRAY_3D(ix, iy, ny, iz, nz) \
00243     (((ix)*(ny) + (iy)) * (nz) + (iz))
00244
00246 #define DEG2DX(dlon, lat) \
00247     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00248
00250 #define DEG2DY(dlat) \
00251     ((dlat) * M_PI * RE / 180.)
00252
00254 #define DP2DZ(dp, p) \
00255     (- (dp) * H0 / (p))
00256
00258 #define DX2DEG(dx, lat) \
00259     ((lat) < -89.999 || (lat) > 89.999) ? 0 \
00260     : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI))
00261
00263 #define DY2DEG(dy) \
00264     ((dy) * 180. / (M_PI * RE))
00265
00267 #define DZ2DP(dz, p) \
00268     (- (dz) * (p) / H0)
00269
00271 #define DIST(a, b) \
00272     sqrt(DIST2(a, b))
00273
00275 #define DIST2(a, b) \
00276     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00277
00279 #define DOTP(a, b) \
00280     (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00281
00283 #define FMOD(x, y) \
00284     ((x) - (int) ((x) / (y)) * (y))
00285
00287 #define FREAD(ptr, type, size, out) { \
00288     if(fread(ptr, sizeof(type), size, out)!=size) \
00289         ERRMSG("Error while reading!"); \
00290 }

```

```

00291
00293 #define FWRITE(ptr, type, size, out) { \
00294     if(fwrite(ptr, sizeof(type), size, out)!=size) \
00295         ERRMSG("Error while writing!"); \
00296 }
00297
00299 #define INTPOL_INIT \
00300     double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};
00301
00303 #define INTPOL_2D(var, init) \
00304     intpol_met_time_2d(met0, met0->var, met1, met1->var, \
00305         atm->time[ip], atm->lon[ip], atm->lat[ip], \
00306         &var, ci, cw, init);
00307
00309 #define INTPOL_3D(var, init) \
00310     intpol_met_time_3d(met0, met0->var, met1, met1->var, \
00311         atm->time[ip], atm->p[ip], \
00312         atm->lon[ip], atm->lat[ip], \
00313         &var, ci, cw, init);
00314
00316 #define INTPOL_SPACE_ALL(p, lon, lat) { \
00317     intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1); \
00318     intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0); \
00319     intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0); \
00320     intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0); \
00321     intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0); \
00322     intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0); \
00323     intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0); \
00324     intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0); \
00325     intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0); \
00326     intpol_met_space_3d(met, met->iwc, p, lon, lat, &iwc, ci, cw, 0); \
00327     intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0); \
00328     intpol_met_space_2d(met, met->ts, lon, lat, &ts, ci, cw, 0); \
00329     intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0); \
00330     intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0); \
00331     intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0); \
00332     intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0); \
00333     intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0); \
00334     intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0); \
00335     intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0); \
00336     intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0); \
00337     intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0); \
00338     intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0); \
00339     intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0); \
00340     intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0); \
00341     intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0); \
00342     intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0); \
00343     intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0); \
00344     intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0); \
00345 }
00346
00348 #define INTPOL_TIME_ALL(time, p, lon, lat) { \
00349     intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
00350     intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
00351     intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
00352     intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
00353     intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
00354     intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
00355     intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
00356     intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
00357     intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
00358     intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
00359     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
00360     intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
00361     intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
00362     intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
00363     intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
00364     intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0); \
00365     intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
00366     intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
00367     intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
00368     intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
00369     intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0); \
00370     intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0); \
00371     intpol_met_time_2d(met0, met0->cl, met1, met1->cl, time, lon, lat, &cl, ci, cw, 0); \
00372     intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
00373     intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
00374     intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
00375     intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
00376     intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0); \
00377 }
00378
00380 #define LAPSE(p1, t1, p2, t2) \
00381     (1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1)) \
00382     * ((p2) + (p1)) / ((p2) - (p1)))
00383
00385 #define LIN(x0, y0, x1, y1, x) \

```

```

00386     ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00387
00389 #define NC(cmd) {
00390     int nc_result=(cmd);
00391     if(nc_result!=NC_NOERR)
00392         ERRMSG("%s", nc_strerror(nc_result));
00393 }
00394
00396 #define NC_DEF_VAR(varname, type, ndims, dims, long_name, units) {
00397     NC(nc_def_var(ncid, varname, type, ndims, dims, &varid));
00398     NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));
00399     NC(nc_put_att_text(ncid, varid, "units", strlen(units), units));
00400 }
00401
00403 #define NC_GET_DOUBLE(varname, ptr, force) {
00404     if(force) {
00405         NC(nc_inq_varid(ncid, varname, &varid));
00406         NC(nc_get_var_double(ncid, varid, ptr));
00407     } else {
00408         if(nc_inq_varid(ncid, varname, &varid) == NC_NOERR) {
00409             NC(nc_get_var_double(ncid, varid, ptr));
00410         } else
00411             WARN("netCDF variable %s is missing!", varname);
00412     }
00413 }
00414
00416 #define NC_INQ_DIM(dimname, ptr, min, max) {
00417     int dimid; size_t naux;
00418     NC(nc_inq_dimid(ncid, dimname, &dimid));
00419     NC(nc_inq_dimlen(ncid, dimid, &naux));
00420     *ptr = (int)naux;
00421     if ((*ptr) < (min) || (*ptr) > (max))
00422         ERRMSG("Dimension %s is out of range!", dimname);
00423 }
00424
00426 #define NC_PUT_DOUBLE(varname, ptr, hyperslab) {
00427     NC(nc_inq_varid(ncid, varname, &varid));
00428     if(hyperslab) {
00429         NC(nc_put_vara_double(ncid, varid, start, count, ptr));
00430     } else {
00431         NC(nc_put_var_double(ncid, varid, ptr));
00432     }
00433 }
00434
00436 #define NC_PUT_INT(varname, ptr, hyperslab) {
00437     NC(nc_inq_varid(ncid, varname, &varid));
00438     if(hyperslab) {
00439         NC(nc_put_vara_int(ncid, varid, start, count, ptr));
00440     } else {
00441         NC(nc_put_var_int(ncid, varid, ptr));
00442     }
00443 }
00444
00446 #define NC_PUT_ATT(attname, text)
00447     NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));
00448
00450 #define NC_PUT_FLOAT(varname, ptr, hyperslab) {
00451     NC(nc_inq_varid(ncid, varname, &varid));
00452     if(hyperslab) {
00453         NC(nc_put_vara_float(ncid, varid, start, count, ptr));
00454     } else {
00455         NC(nc_put_var_float(ncid, varid, ptr));
00456     }
00457 }
00458
00460 #define NN(x0, y0, x1, y1, x)
00461     (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))
00462
00464 #define NORM(a)
00465     sqrt(DOTP(a, a))
00466
00468 #define P(z)
00469     (P0 * exp(-(z) / H0))
00470
00472 #define PSAT(t)
00473     (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))
00474
00476 #define PSICE(t)
00477     (0.01 * pow(10., -2663.5 / (t) + 12.537))
00478
00480 #define PW(p, h2o)
00481     ((p) * GSL_MAX((h2o), 0.1e-6)
00482      / (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
00483
00485 #define RH(p, t, h2o)
00486     (PW(p, h2o) / PSAT(t) * 100.)
00487

```

```

00489 #define RHICE(p, t, h2o) \
00490     (PW(p, h2o) / PSICE(t) * 100.)
00491
00493 #define RHO(p, t) \
00494     (100. * (p) / (RA * (t)))
00495
00497 #define SET_ATM(qnt, val) \
00498     if (ctl->qnt >= 0) \
00499         atm->q[ctl->qnt][ip] = val;
00500
00502 #define SET_QNT(qnt, name, longname, unit) \
00503     if (strcasecmp(ctl->qnt_name[iq], name) == 0) { \
00504         ctl->qnt = iq; \
00505         sprintf(ctl->qnt_longname[iq], longname); \
00506         sprintf(ctl->qnt_unit[iq], unit); \
00507     } else
00508
00510 #define SH(h2o) \
00511     (EPS * GSL_MAX((h2o), 0.1e-6))
00512
00514 #define SQR(x) \
00515     ((x)*(x))
00516
00518 #define SWAP(x, y, type) \
00519     do {type tmp = x; x = y; y = tmp;} while(0);
00520
00522 #define TDEW(p, h2o) \
00523     (T0 + 243.12 * log(PW((p), (h2o)) / 6.112) \
00524     / (17.62 - log(PW((p), (h2o)) / 6.112)))
00525
00527 #define TICE(p, h2o) \
00528     (-2663.5 / (log10(100. * PW((p), (h2o)))) - 12.537))
00529
00531 #define THETA(p, t) \
00532     ((t) * pow(1000. / (p), 0.286))
00533
00535 #define THETAVIRT(p, t, h2o) \
00536     (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))
00537
00539 #define TOK(line, tok, format, var) { \
00540     if((tok)=strtok((line), " \t")) { \
00541         if(sscanf(tok, format, &(var))!=1) continue; \
00542     } else ERRMSG("Error while reading!"); \
00543 }
00544
00546 #define TVIRT(t, h2o) \
00547     ((t) * (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
00548
00550 #define Z(p) \
00551     (H0 * log(P0 / (p)))
00552
00554 #define ZDIFF(lnp0, t0, h2o0, lnp1, t1, h2o1) \
00555     (RI / MA / G0 * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1))) \
00556     * ((lnp0) - (lnp1)))
00557
00559 #define ZETA(ps, p, t) \
00560     ((p) / (ps) <= 0.3 ? 1. : \
00561     sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3))) \
00562     * THETA((p), (t)))
00563
00564 /* -----
00565     Log messages...
00566     ----- */
00567
00569 #ifndef LOGLEV
00570 #define LOGLEV 2
00571 #endif
00572
00574 #define LOG(level, ...) { \
00575     if(level >= 2) \
00576         printf(" "); \
00577     if(level <= LOGLEV) { \
00578         printf(__VA_ARGS__); \
00579         printf("\n"); \
00580     } \
00581 }
00582
00584 #define WARN(...) { \
00585     printf("\nWarning (%s, %s, %d): ", __FILE__, __func__, __LINE__); \
00586     LOG(0, __VA_ARGS__); \
00587 }
00588
00590 #define ERRMSG(...) { \
00591     printf("\nError (%s, %s, %d): ", __FILE__, __func__, __LINE__); \
00592     LOG(0, __VA_ARGS__); \
00593     exit(EXIT_FAILURE); \
00594 }

```

```

00595
00597 #define PRINT(format, var) \
00598     printf("Print (%s, %s, l%d): %s= "format"\n", \
00599         __FILE__, __func__, __LINE__, #var, var);
00600
00601 /* -----
00602     Timers...
00603     ----- */
00604
00606 #define NTIMER 100
00607
00609 #define PRINT_TIMERS \
00610     timer("END", "END", 1);
00611
00613 #define SELECT_TIMER(id, group, color) { \
00614     NVTX_POP; \
00615     NVTX_PUSH(id, color); \
00616     timer(id, group, 0); \
00617 }
00618
00620 #define START_TIMERS \
00621     NVTX_PUSH("START", NVTX_CPU);
00622
00624 #define STOP_TIMERS \
00625     NVTX_POP;
00626
00627 /* -----
00628     NVIDIA Tools Extension (NVTX)...
00629     ----- */
00630
00631 #ifdef NVTX
00632 #include "nvToolsExt.h"
00633
00635 #define NVTX_CPU 0xFFADD8E6
00636
00638 #define NVTX_GPU 0xFF00008B
00639
00641 #define NVTX_H2D 0xFFFFFFFF00
00642
00644 #define NVTX_D2H 0xFFFF8800
00645
00647 #define NVTX_READ 0xFFFFCCCB
00648
00650 #define NVTX_WRITE 0xFF8B0000
00651
00653 #define NVTX_PUSH(range_title, range_color) { \
00654     nvtxEvtAttributes_t eventAttrib = {0}; \
00655     eventAttrib.version = NVTX_VERSION; \
00656     eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE; \
00657     eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII; \
00658     eventAttrib.colorType = NVTX_COLOR_ARGB; \
00659     eventAttrib.color = range_color; \
00660     eventAttrib.message.ascii = range_title; \
00661     nvtxRangePushEx(&eventAttrib); \
00662 }
00663
00665 #define NVTX_POP { \
00666     nvtxRangePop(); \
00667 } \
00668 #else
00669
00670 /* Empty definitions of NVTX_PUSH and NVTX_POP... */
00671 #define NVTX_PUSH(range_title, range_color) {}
00672 #define NVTX_POP {}
00673 #endif
00674
00675 /* -----
00676     Thrust...
00677     ----- */
00678
00680 void thrustSortWrapper(
00681     double *__restrict__ c,
00682     int n,
00683     int *__restrict__ index);
00684
00685 /* -----
00686     Structs...
00687     ----- */
00688
00690 typedef struct {
00691
00693     int vert_coord_ap;
00694
00696     int vert_coord_met;
00697
00699     int vert_vel;
00700

```



```
00702 int clams_met_data;
00703
00705 size_t chunkszhint;
00706
00708 int read_mode;
00709
00711 int nq;
00712
00714 char qnt_name[NQ][LEN];
00715
00717 char qnt_longname[NQ][LEN];
00718
00720 char qnt_unit[NQ][LEN];
00721
00723 char qnt_format[NQ][LEN];
00724
00726 int qnt_idx;
00727
00729 int qnt_ens;
00730
00732 int qnt_stat;
00733
00735 int qnt_m;
00736
00738 int qnt_vmr;
00739
00741 int qnt_rp;
00742
00744 int qnt_rhop;
00745
00747 int qnt_ps;
00748
00750 int qnt_ts;
00751
00753 int qnt_zs;
00754
00756 int qnt_us;
00757
00759 int qnt_vs;
00760
00762 int qnt_pbl;
00763
00765 int qnt_pt;
00766
00768 int qnt_tt;
00769
00771 int qnt_zt;
00772
00774 int qnt_h2ot;
00775
00777 int qnt_z;
00778
00780 int qnt_p;
00781
00783 int qnt_t;
00784
00786 int qnt_rho;
00787
00789 int qnt_u;
00790
00792 int qnt_v;
00793
00795 int qnt_w;
00796
00798 int qnt_h2o;
00799
00801 int qnt_o3;
00802
00804 int qnt_lwc;
00805
00807 int qnt_iwc;
00808
00810 int qnt_pct;
00811
00813 int qnt_pcb;
00814
00816 int qnt_cl;
00817
00819 int qnt_plcl;
00820
00822 int qnt_plfc;
00823
00825 int qnt_pel;
00826
00828 int qnt_cape;
00829
00831 int qnt_cin;
```

```
00832
00834     int qnt_hno3;
00835
00837     int qnt_oh;
00838
00840     int qnt_vmrimpl;
00841
00843     int qnt_mloss_oh;
00844
00846     int qnt_mloss_h2o2;
00847
00849     int qnt_mloss_wet;
00850
00852     int qnt_mloss_dry;
00853
00855     int qnt_mloss_decay;
00856
00858     int qnt_psat;
00859
00861     int qnt_psice;
00862
00864     int qnt_pw;
00865
00867     int qnt_sh;
00868
00870     int qnt_rh;
00871
00873     int qnt_rhice;
00874
00876     int qnt_theta;
00877
00879     int qnt_zeta;
00880
00882     int qnt_tvirt;
00883
00885     int qnt_lapse;
00886
00888     int qnt_vh;
00889
00891     int qnt_vz;
00892
00894     int qnt_pv;
00895
00897     int qnt_tdew;
00898
00900     int qnt_tice;
00901
00903     int qnt_tsts;
00904
00906     int qnt_tnat;
00907
00909     int direction;
00910
00912     double t_start;
00913
00915     double t_stop;
00916
00918     double dt_mod;
00919
00921     char metbase[LEN];
00922
00924     double dt_met;
00925
00927     int met_type;
00928
00930     int met_nc_scale;
00931
00933     int met_dx;
00934
00936     int met_dy;
00937
00939     int met_dp;
00940
00942     int met_sx;
00943
00945     int met_sy;
00946
00948     int met_sp;
00949
00951     double met_detrend;
00952
00954     int met_np;
00955
00957     double met_p[EP];
00958
00960     int met_geopot_sx;
00961
```

```
00963 int met_geopot_sy;
00964
00967 int met_tropo;
00968
00970 double met_tropo_lapse;
00971
00973 int met_tropo_nlev;
00974
00976 double met_tropo_lapse_sep;
00977
00979 int met_tropo_nlev_sep;
00980
00982 double met_tropo_pv;
00983
00985 double met_tropo_theta;
00986
00988 int met_tropo_spline;
00989
00991 int met_cloud;
00992
00994 double met_cloud_min;
00995
00997 double met_dt_out;
00998
01000 int met_cache;
01001
01003 double sort_dt;
01004
01007 int isosurf;
01008
01010 char balloon[LEN];
01011
01013 int advect;
01014
01016 int reflect;
01017
01019 double turb_dx_trop;
01020
01022 double turb_dx_strat;
01023
01025 double turb_dz_trop;
01026
01028 double turb_dz_strat;
01029
01031 double turb_mesox;
01032
01034 double turb_mesoz;
01035
01037 double conv_cape;
01038
01040 double conv_cin;
01041
01043 double conv_wmax;
01044
01046 double conv_wcape;
01047
01049 double conv_dt;
01050
01052 int conv_mix;
01053
01055 int conv_mix_bot;
01056
01058 int conv_mix_top;
01059
01061 double bound_mass;
01062
01064 double bound_mass_trend;
01065
01067 double bound_vmr;
01068
01070 double bound_vmr_trend;
01071
01073 double bound_lat0;
01074
01076 double bound_lat1;
01077
01079 double bound_p0;
01080
01082 double bound_p1;
01083
01085 double bound_dps;
01086
01088 char species[LEN];
01089
01091 double molmass;
01092
01094 double tdec_trop;
```

```
01095
01097 double tdec_strat;
01098
01100 char clim_oh_filename[LEN];
01101
01103 char clim_h2o2_filename[LEN];
01104
01106 int oh_chem_reaction;
01107
01109 double oh_chem[4];
01110
01112 double oh_chem_beta;
01113
01115 double h2o2_chem_cc;
01116
01118 int h2o2_chem_reaction;
01119
01121 double dry_depo[1];
01122
01124 double wet_depo_pre[2];
01125
01127 double wet_depo_bc_a;
01128
01130 double wet_depo_bc_b;
01131
01133 double wet_depo_ic_a;
01134
01136 double wet_depo_ic_b;
01137
01139 double wet_depo_ic_h[3];
01140
01142 double wet_depo_bc_h[2];
01143
01145 double wet_depo_ic_ret_ratio;
01146
01148 double wet_depo_bc_ret_ratio;
01149
01151 double psc_h2o;
01152
01154 double psc_hno3;
01155
01157 char atm_basename[LEN];
01158
01160 char atm_gpfile[LEN];
01161
01163 double atm_dt_out;
01164
01166 int atm_filter;
01167
01169 int atm_stride;
01170
01172 int atm_type;
01173
01175 char csi_basename[LEN];
01176
01178 double csi_dt_out;
01179
01181 char csi_obsfile[LEN];
01182
01184 double csi_obsmin;
01185
01187 double csi_modmin;
01188
01190 int csi_nz;
01191
01193 double csi_z0;
01194
01196 double csi_z1;
01197
01199 int csi_nx;
01200
01202 double csi_lon0;
01203
01205 double csi_lon1;
01206
01208 int csi_ny;
01209
01211 double csi_lat0;
01212
01214 double csi_lat1;
01215
01217 char ens_basename[LEN];
01218
01220 double ens_dt_out;
01221
01223 char grid_basename[LEN];
01224
```

```
01226 char grid_gpfile[LEN];
01227
01229 double grid_dt_out;
01230
01232 int grid_sparse;
01233
01235 int grid_nz;
01236
01238 double grid_z0;
01239
01241 double grid_z1;
01242
01244 int grid_nx;
01245
01247 double grid_lon0;
01248
01250 double grid_lon1;
01251
01253 int grid_ny;
01254
01256 double grid_lat0;
01257
01259 double grid_lat1;
01260
01262 int grid_type;
01263
01265 char prof_basename[LEN];
01266
01268 char prof_obsfile[LEN];
01269
01271 int prof_nz;
01272
01274 double prof_z0;
01275
01277 double prof_z1;
01278
01280 int prof_nx;
01281
01283 double prof_lon0;
01284
01286 double prof_lon1;
01287
01289 int prof_ny;
01290
01292 double prof_lat0;
01293
01295 double prof_lat1;
01296
01298 char sample_basename[LEN];
01299
01301 char sample_obsfile[LEN];
01302
01304 double sample_dx;
01305
01307 double sample_dz;
01308
01310 char stat_basename[LEN];
01311
01313 double stat_lon;
01314
01316 double stat_lat;
01317
01319 double stat_r;
01320
01322 double stat_t0;
01323
01325 double stat_t1;
01326
01327 } ctl_t;
01328
01330 typedef struct {
01331
01333 int np;
01334
01336 double time[NP];
01337
01339 double p[NP];
01340
01342 double zeta[NP];
01343
01345 double lon[NP];
01346
01348 double lat[NP];
01349
01351 double q[NQ][NP];
01352
01353 } atm_t;
```

```
01354
01356 typedef struct {
01357
01359     double iso_var[NP];
01360
01362     double iso_ps[NP];
01363
01365     double iso_ts[NP];
01366
01368     int iso_n;
01369
01371     float uvwp[NP][3];
01372
01373 } cache_t;
01374
01376 typedef struct {
01377
01379     int tropo_ntime;
01380
01382     int tropo_nlat;
01383
01385     double tropo_time[12];
01386
01388     double tropo_lat[73];
01389
01391     double tropo[12][73];
01392
01394     int hno3_ntime;
01395
01397     int hno3_nlat;
01398
01400     int hno3_np;
01401
01403     double hno3_time[12];
01404
01406     double hno3_lat[18];
01407
01409     double hno3_p[10];
01410
01412     double hno3[12][18][10];
01413
01415     int oh_ntime;
01416
01418     int oh_nlat;
01419
01421     int oh_np;
01422
01424     double oh_time[CT];
01425
01427     double oh_lat[CY];
01428
01430     double oh_p[CP];
01431
01433     double oh[CT][CP][CY];
01434
01436     int h2o2_ntime;
01437
01439     int h2o2_nlat;
01440
01442     int h2o2_np;
01443
01445     double h2o2_time[CT];
01446
01448     double h2o2_lat[CY];
01449
01451     double h2o2_p[CP];
01452
01454     double h2o2[CT][CP][CY];
01455
01456 } clim_t;
01457
01459 typedef struct {
01460
01462     double time;
01463
01465     int nx;
01466
01468     int ny;
01469
01471     int np;
01472
01474     double lon[EX];
01475
01477     double lat[EY];
01478
01480     double p[EP];
01481
```

```

01483 float ps[EX][EY];
01484
01486 float ts[EX][EY];
01487
01489 float zs[EX][EY];
01490
01492 float us[EX][EY];
01493
01495 float vs[EX][EY];
01496
01498 float pbl[EX][EY];
01499
01501 float pt[EX][EY];
01502
01504 float tt[EX][EY];
01505
01507 float zt[EX][EY];
01508
01510 float h2ot[EX][EY];
01511
01513 float pct[EX][EY];
01514
01516 float pcb[EX][EY];
01517
01519 float cl[EX][EY];
01520
01522 float plcl[EX][EY];
01523
01525 float plfc[EX][EY];
01526
01528 float pel[EX][EY];
01529
01531 float cape[EX][EY];
01532
01534 float cin[EX][EY];
01535
01537 float z[EX][EY][EP];
01538
01540 float t[EX][EY][EP];
01541
01543 float u[EX][EY][EP];
01544
01546 float v[EX][EY][EP];
01547
01549 float w[EX][EY][EP];
01550
01552 float pv[EX][EY][EP];
01553
01555 float h2o[EX][EY][EP];
01556
01558 float o3[EX][EY][EP];
01559
01561 float lwc[EX][EY][EP];
01562
01564 float iwc[EX][EY][EP];
01565
01567 float pl[EX][EY][EP];
01568
01570 float patp[EX][EY][EP];
01571
01573 float zeta[EX][EY][EP];
01574
01576 float zeta_dot[EX][EY][EP];
01577
01578 #ifdef UVW
01580 float uvw[EX][EY][EP][3];
01581 #endif
01582
01583 } met_t;
01584
01585 /* -----
01586 Functions...
01587 ----- */
01588
01590 void cart2geo(
01591     double *x,
01592     double *z,
01593     double *lon,
01594     double *lat);
01595
01597 #ifdef _OPENACC
01598 #pragma acc routine (check_finite)
01599 #endif
01600 int check_finite(
01601     const double x);
01602
01604 #ifdef _OPENACC

```

```
01605 #pragma acc routine (clim_hno3)
01606 #endif
01607 double clim_hno3(
01608     clim_t * clim,
01609     double t,
01610     double lat,
01611     double p);
01612
01614 void clim_hno3_init(
01615     clim_t * clim);
01616
01618 #ifdef _OPENACC
01619 #pragma acc routine (clim_oh)
01620 #endif
01621 double clim_oh(
01622     clim_t * clim,
01623     double t,
01624     double lat,
01625     double p);
01626
01628 #ifdef _OPENACC
01629 #pragma acc routine (clim_oh_diurnal)
01630 #endif
01631 double clim_oh_diurnal(
01632     ctl_t * ctl,
01633     clim_t * clim,
01634     double t,
01635     double p,
01636     double lon,
01637     double lat);
01638
01640 void clim_oh_init(
01641     ctl_t * ctl,
01642     clim_t * clim);
01643
01645 double clim_oh_init_help(
01646     double beta,
01647     double time,
01648     double lat);
01649
01651 #ifdef _OPENACC
01652 #pragma acc routine (clim_h2o2)
01653 #endif
01654 double clim_h2o2(
01655     clim_t * clim,
01656     double t,
01657     double lat,
01658     double p);
01659
01661 void clim_h2o2_init(
01662     ctl_t * ctl,
01663     clim_t * clim);
01664
01666 #ifdef _OPENACC
01667 #pragma acc routine (clim_tropo)
01668 #endif
01669 double clim_tropo(
01670     clim_t * clim,
01671     double t,
01672     double lat);
01673
01675 void clim_tropo_init(
01676     clim_t * clim);
01677
01679 void compress_pack(
01680     char *varname,
01681     float *array,
01682     size_t nxy,
01683     size_t nz,
01684     int decompress,
01685     FILE * inout);
01686
01688 #ifdef ZFP
01689 void compress_zfp(
01690     char *varname,
01691     float *array,
01692     int nx,
01693     int ny,
01694     int nz,
01695     int precision,
01696     double tolerance,
01697     int decompress,
01698     FILE * inout);
01699 #endif
01700
01702 #ifdef ZSTD
01703 void compress_zstd(
```



```

01704     char *varname,
01705     float *array,
01706     size_t n,
01707     int decompress,
01708     FILE * inout);
01709 #endif
01710
01712 void day2doy(
01713     int year,
01714     int mon,
01715     int day,
01716     int *doy);
01717
01719 void doy2day(
01720     int year,
01721     int doy,
01722     int *mon,
01723     int *day);
01724
01726 void geo2cart(
01727     double z,
01728     double lon,
01729     double lat,
01730     double *x);
01731
01733 void get_met(
01734     ctl_t * ctl,
01735     clim_t * clim,
01736     double t,
01737     met_t ** met0,
01738     met_t ** met1);
01739
01741 void get_met_help(
01742     ctl_t * ctl,
01743     double t,
01744     int direct,
01745     char *metbase,
01746     double dt_met,
01747     char *filename);
01748
01750 void get_met_replace(
01751     char *orig,
01752     char *search,
01753     char *repl);
01754
01756 #ifdef _OPENACC
01757 #pragma acc routine (intpol_met_space_3d)
01758 #endif
01759 void intpol_met_space_3d(
01760     met_t * met,
01761     float array[EX][EY][EP],
01762     double p,
01763     double lon,
01764     double lat,
01765     double *var,
01766     int *ci,
01767     double *cw,
01768     int init);
01769
01771 #ifdef _OPENACC
01772 #pragma acc routine (intpol_met_space_2d)
01773 #endif
01774 void intpol_met_space_2d(
01775     met_t * met,
01776     float array[EX][EY],
01777     double lon,
01778     double lat,
01779     double *var,
01780     int *ci,
01781     double *cw,
01782     int init);
01783
01785 #ifdef UVW
01786 #ifdef _OPENACC
01787 #pragma acc routine (intpol_met_space_uvw)
01788 #endif
01789 void intpol_met_space_uvw(
01790     met_t * met,
01791     double p,
01792     double lon,
01793     double lat,
01794     double *u,
01795     double *v,
01796     double *w,
01797     int *ci,
01798     double *cw,
01799     int init);

```

```
01800 #endif
01801
01803 #ifdef _OPENACC
01804 #pragma acc routine (intpol_met_time_3d)
01805 #endif
01806 void intpol_met_time_3d(
01807     met_t * met0,
01808     float array0[EX][EY][EP],
01809     met_t * met1,
01810     float array1[EX][EY][EP],
01811     double ts,
01812     double p,
01813     double lon,
01814     double lat,
01815     double *var,
01816     int *ci,
01817     double *cw,
01818     int init);
01819
01821 #ifdef _OPENACC
01822 #pragma acc routine (intpol_met_time_2d)
01823 #endif
01824 void intpol_met_time_2d(
01825     met_t * met0,
01826     float array0[EX][EY],
01827     met_t * met1,
01828     float array1[EX][EY],
01829     double ts,
01830     double lon,
01831     double lat,
01832     double *var,
01833     int *ci,
01834     double *cw,
01835     int init);
01836
01838 #ifdef UVW
01839 #ifdef _OPENACC
01840 #pragma acc routine (intpol_met_time_uvw)
01841 #endif
01842 void intpol_met_time_uvw(
01843     met_t * met0,
01844     met_t * met1,
01845     double ts,
01846     double p,
01847     double lon,
01848     double lat,
01849     double *u,
01850     double *v,
01851     double *w);
01852 #endif
01853
01855 void jsec2time(
01856     double jsec,
01857     int *year,
01858     int *mon,
01859     int *day,
01860     int *hour,
01861     int *min,
01862     int *sec,
01863     double *remain);
01864
01866 #ifdef _OPENACC
01867 #pragma acc routine (lapse_rate)
01868 #endif
01869 double lapse_rate(
01870     double t,
01871     double h2o);
01872
01874 #ifdef _OPENACC
01875 #pragma acc routine (locate_irr)
01876 #endif
01877 int locate_irr(
01878     double **x,
01879     int n,
01880     double x);
01881
01883 #ifdef _OPENACC
01884 #pragma acc routine (locate_reg)
01885 #endif
01886 int locate_reg(
01887     double **x,
01888     int n,
01889     double x);
01890
01892 #ifdef _OPENACC
01893 #pragma acc routine (nat_temperature)
01894 #endif
```

```
01895 double nat_temperature(  
01896     double p,  
01897     double h2o,  
01898     double hno3);  
01899  
01901 void quicksort(  
01902     double arr[],  
01903     int brr[],  
01904     int low,  
01905     int high);  
01906  
01908 int quicksort_partition(  
01909     double arr[],  
01910     int brr[],  
01911     int low,  
01912     int high);  
01913  
01915 int read_atm(  
01916     const char *filename,  
01917     ctl_t * ctl,  
01918     atm_t * atm);  
01919  
01921 int read_atm_asc(  
01922     const char *filename,  
01923     ctl_t * ctl,  
01924     atm_t * atm);  
01925  
01927 int read_atm_bin(  
01928     const char *filename,  
01929     ctl_t * ctl,  
01930     atm_t * atm);  
01931  
01933 int read_atm_clams(  
01934     const char *filename,  
01935     ctl_t * ctl,  
01936     atm_t * atm);  
01937  
01939 int read_atm_nc(  
01940     const char *filename,  
01941     ctl_t * ctl,  
01942     atm_t * atm);  
01943  
01945 void read_clim(  
01946     ctl_t * ctl,  
01947     clim_t * clim);  
01948  
01950 void read_ctl(  
01951     const char *filename,  
01952     int argc,  
01953     char *argv[],  
01954     ctl_t * ctl);  
01955  
01957 int read_met(  
01958     char *filename,  
01959     ctl_t * ctl,  
01960     clim_t * clim,  
01961     met_t * met);  
01962  
01964 void read_met_bin_2d(  
01965     FILE * out,  
01966     met_t * met,  
01967     float var[EX][EY],  
01968     char *varname);  
01969  
01971 void read_met_bin_3d(  
01972     FILE * in,  
01973     ctl_t * ctl,  
01974     met_t * met,  
01975     float var[EX][EY][EP],  
01976     char *varname,  
01977     int precision,  
01978     double tolerance);  
01979  
01981 void read_met_cape(  
01982     clim_t * clim,  
01983     met_t * met);  
01984  
01986 void read_met_cloud(  
01987     ctl_t * ctl,  
01988     met_t * met);  
01989  
01991 void read_met_detrend(  
01992     ctl_t * ctl,  
01993     met_t * met);  
01994  
01996 void read_met_extrapolate(  
01997     met_t * met);
```

```
01998
02000 void read_met_geopot(
02001     ctl_t * ctl,
02002     met_t * met);
02003
02005 void read_met_grid(
02006     char *filename,
02007     int ncid,
02008     ctl_t * ctl,
02009     met_t * met);
02010
02012 void read_met_levels(
02013     int ncid,
02014     ctl_t * ctl,
02015     met_t * met);
02016
02018 void read_met_m12p1(
02019     ctl_t * ctl,
02020     met_t * met,
02021     float var[EX][EY][EP]);
02022
02024 int read_met_nc_2d(
02025     int ncid,
02026     char *varname,
02027     char *varname2,
02028     ctl_t * ctl,
02029     met_t * met,
02030     float dest[EX][EY],
02031     float scl,
02032     int init);
02033
02035 int read_met_nc_3d(
02036     int ncid,
02037     char *varname,
02038     char *varname2,
02039     ctl_t * ctl,
02040     met_t * met,
02041     float dest[EX][EY][EP],
02042     float scl,
02043     int init);
02044
02046 void read_met_pbl(
02047     met_t * met);
02048
02050 void read_met_periodic(
02051     met_t * met);
02052
02054 void read_met_pv(
02055     met_t * met);
02056
02058 void read_met_sample(
02059     ctl_t * ctl,
02060     met_t * met);
02061
02063 void read_met_surface(
02064     int ncid,
02065     met_t * met,
02066     ctl_t * ctl);
02067
02069 void read_met_tropo(
02070     ctl_t * ctl,
02071     clim_t * clim,
02072     met_t * met);
02073
02075 void read_obs(
02076     char *filename,
02077     double *rt,
02078     double *rz,
02079     double *rlon,
02080     double *rlat,
02081     double *robs,
02082     int *nobs);
02083
02085 double scan_ctl(
02086     const char *filename,
02087     int argc,
02088     char *argv[],
02089     const char *varname,
02090     int arridx,
02091     const char *defvalue,
02092     char *value);
02093
02095 #ifdef _OPENACC
02096 #pragma acc routine (sedi)
02097 #endif
02098 double sedi(
02099     double p,
```

```
02100     double T,
02101     double rp,
02102     double rhop);
02103
02105 void spline(
02106     double *x,
02107     double *y,
02108     int n,
02109     double *x2,
02110     double *y2,
02111     int n2,
02112     int method);
02113
02115 #ifdef _OPENACC
02116 #pragma acc routine (stddev)
02117 #endif
02118 float stddev(
02119     float *data,
02120     int n);
02121
02123 #ifdef _OPENACC
02124 #pragma acc routine (sza)
02125 #endif
02126 double sza(
02127     double sec,
02128     double lon,
02129     double lat);
02130
02132 void time2jsec(
02133     int year,
02134     int mon,
02135     int day,
02136     int hour,
02137     int min,
02138     int sec,
02139     double remain,
02140     double *jsec);
02141
02143 void timer(
02144     const char *name,
02145     const char *group,
02146     int output);
02147
02149 #ifdef _OPENACC
02150 #pragma acc routine (tropo_weight)
02151 #endif
02152 double tropo_weight(
02153     clim_t * clim,
02154     double t,
02155     double lat,
02156     double p);
02157
02159 void write_atm(
02160     const char *filename,
02161     ctl_t * ctl,
02162     atm_t * atm,
02163     double t);
02164
02166 void write_atm_asc(
02167     const char *filename,
02168     ctl_t * ctl,
02169     atm_t * atm,
02170     double t);
02171
02173 void write_atm_bin(
02174     const char *filename,
02175     ctl_t * ctl,
02176     atm_t * atm);
02177
02179 void write_atm_clams(
02180     ctl_t * ctl,
02181     atm_t * atm,
02182     double t);
02183
02185 void write_atm_nc(
02186     const char *filename,
02187     ctl_t * ctl,
02188     atm_t * atm);
02189
02191 void write_csi(
02192     const char *filename,
02193     ctl_t * ctl,
02194     atm_t * atm,
02195     double t);
02196
02198 void write_ens(
02199     const char *filename,
```

```
02200     ctl_t * ctl,
02201     atm_t * atm,
02202     double t);
02203
02205 void write_grid(
02206     const char *filename,
02207     ctl_t * ctl,
02208     met_t * met0,
02209     met_t * met1,
02210     atm_t * atm,
02211     double t);
02212
02214 void write_grid_asc(
02215     const char *filename,
02216     ctl_t * ctl,
02217     double *cd,
02218     double *vmr_expl,
02219     double *vmr_impl,
02220     double t,
02221     double *z,
02222     double *lon,
02223     double *lat,
02224     double *area,
02225     double dz,
02226     int *np);
02227
02229 void write_grid_nc(
02230     const char *filename,
02231     ctl_t * ctl,
02232     double *cd,
02233     double *vmr_expl,
02234     double *vmr_impl,
02235     double t,
02236     double *z,
02237     double *lon,
02238     double *lat,
02239     double *area,
02240     double dz,
02241     int *np);
02242
02244 int write_met(
02245     char *filename,
02246     ctl_t * ctl,
02247     met_t * met);
02248
02250 void write_met_bin_2d(
02251     FILE * out,
02252     met_t * met,
02253     float var[EX][EY],
02254     char *varname);
02255
02257 void write_met_bin_3d(
02258     FILE * out,
02259     ctl_t * ctl,
02260     met_t * met,
02261     float var[EX][EY][EP],
02262     char *varname,
02263     int precision,
02264     double tolerance);
02265
02267 void write_prof(
02268     const char *filename,
02269     ctl_t * ctl,
02270     met_t * met0,
02271     met_t * met1,
02272     atm_t * atm,
02273     double t);
02274
02276 void write_sample(
02277     const char *filename,
02278     ctl_t * ctl,
02279     met_t * met0,
02280     met_t * met1,
02281     atm_t * atm,
02282     double t);
02283
02285 void write_station(
02286     const char *filename,
02287     ctl_t * ctl,
02288     atm_t * atm,
02289     double t);
02290
02291 #endif /* LIBTRAC_H */
```

## 5.23 met\_conv.c File Reference

Convert file format of meteo data files.

```
#include "libtrac.h"
```

### Functions

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

#### 5.23.1 Detailed Description

Convert file format of meteo data files.

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

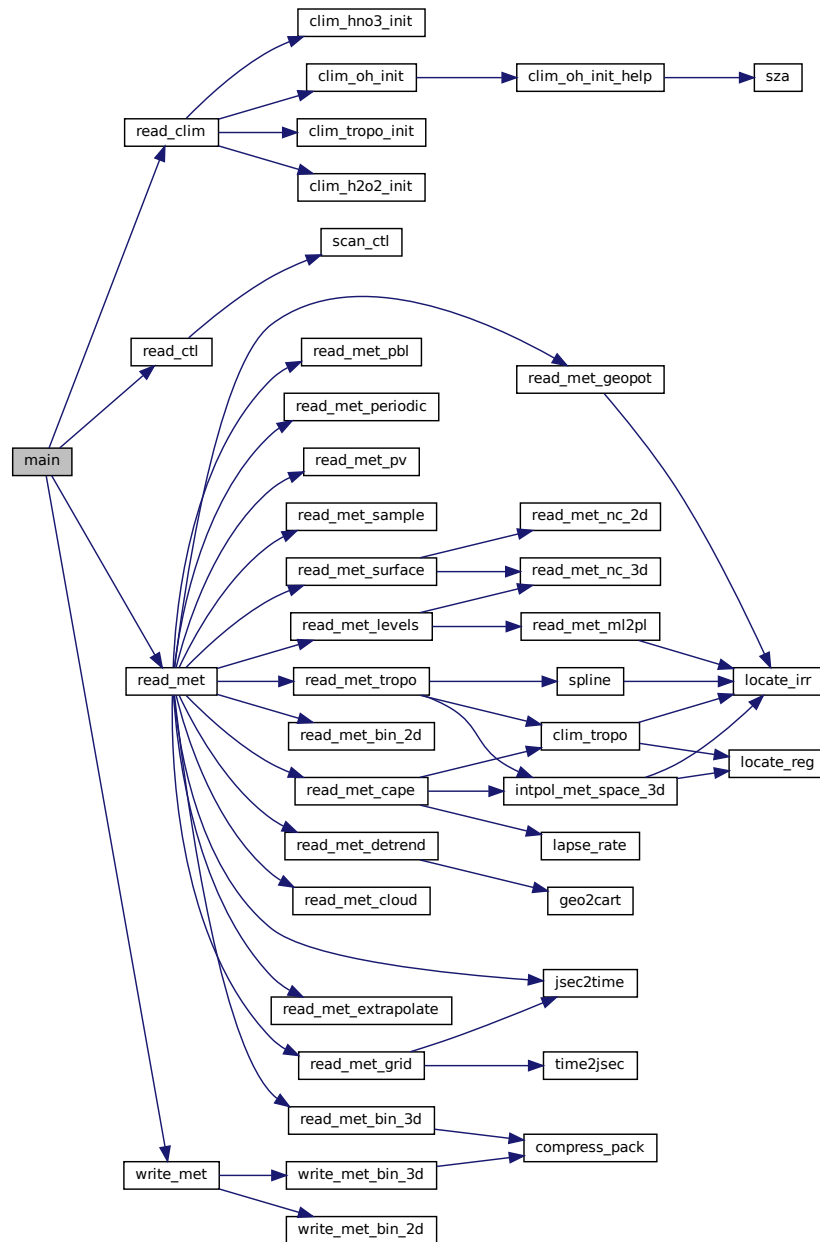
#### 5.23.2 Function Documentation

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

Definition at line 27 of file [met\\_conv.c](#).

```
00029         {
00030
00031     ctl_t ctl;
00032
00033     clim_t *clim;
00034
00035     met_t *met;
00036
00037     /* Check arguments... */
00038     if (argc < 6)
00039         ERRMSG("Give parameters: <ctl> <met_in> <met_in_type>"
00040              " <met_out> <met_out_type>");
00041
00042     /* Allocate... */
00043     ALLOC(clim, clim_t, 1);
00044     ALLOC(met, met_t, 1);
00045
00046     /* Read control parameters... */
00047     read_ctl(argv[1], argc, argv, &ctl);
00048
00049     /* Read climatological data... */
00050     read_clim(&ctl, clim);
00051
00052     /* Read meteo data... */
00053     ctl.met_type = atoi(argv[3]);
00054     if (!read_met(argv[2], &ctl, clim, met))
00055         ERRMSG("Cannot open file!");
00056
00057     /* Write meteo data... */
00058     ctl.met_type = atoi(argv[5]);
00059     write_met(argv[4], &ctl, met);
00060
00061     /* Free... */
00062     free(clim);
00063     free(met);
00064
00065     return EXIT_SUCCESS;
00066 }
```

Here is the call graph for this function:



## 5.24 met\_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) 2013-2022 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      clim_t *clim;
00034
00035      met_t *met;
00036
00037      /* Check arguments... */
00038      if (argc < 6)
00039          ERRMSG("Give parameters: <ctl> <met_in> <met_in_type>"
00040                " <met_out> <met_out_type>");
00041
00042      /* Allocate... */
00043      ALLOC(clim, clim_t, 1);
00044      ALLOC(met, met_t, 1);
00045
00046      /* Read control parameters... */
00047      read_ctl(argv[1], argc, argv, &ctl);
00048
00049      /* Read climatological data... */
00050      read_clim(&ctl, clim);
00051
00052      /* Read meteo data... */
00053      ctl.met_type = atoi(argv[3]);
00054      if (!read_met(argv[2], &ctl, clim, met))
00055          ERRMSG("Cannot open file!");
00056
00057      /* Write meteo data... */
00058      ctl.met_type = atoi(argv[5]);
00059      write_met(argv[4], &ctl, met);
00060
00061      /* Free... */
00062      free(clim);
00063      free(met);
00064
00065      return EXIT_SUCCESS;
00066 }

```

## 5.25 met\_lapse.c File Reference

Calculate lapse rate statistics.

```
#include "libtrac.h"
```

### Macros

- `#define LAPSEMIN -20.0`  
*Lapse rate minimum [K/km].*
- `#define DLAPSE 0.1`  
*Lapse rate bin size [K/km].*
- `#define IDXMAX 400`  
*Maximum number of histogram bins.*

### Functions

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

### 5.25.1 Detailed Description

Calculate lapse rate statistics.

Definition in file [met\\_lapse.c](#).

### 5.25.2 Macro Definition Documentation

#### 5.25.2.1 LAPSEMIN `#define LAPSEMIN -20.0`

Lapse rate minimum [K/km].

Definition at line 32 of file [met\\_lapse.c](#).

#### 5.25.2.2 DLAPSE `#define DLAPSE 0.1`

Lapse rate bin size [K/km].

Definition at line 35 of file [met\\_lapse.c](#).

#### 5.25.2.3 IDXMAX `#define IDXMAX 400`

Maximum number of histogram bins.

Definition at line 38 of file [met\\_lapse.c](#).

### 5.25.3 Function Documentation

```

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

```

Definition at line 44 of file [met\\_lapse.c](#).

```

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

```

```

00126
00127     /* Check surface pressure... */
00128     if (p2[iz] > met->ps[ix][iy])
00129         continue;
00130
00131     /* Get mean latitude and height... */
00132     lat_mean += met->lat[iy];
00133     z_mean += z2[iz];
00134     np++;
00135
00136     /* Get lapse rates within a vertical layer... */
00137     int nlapse = 0;
00138     double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00139         0, lapse_sig = 0;
00140     for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00141         lapse_max =
00142             GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00143         lapse_min =
00144             GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00145         lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00146         lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00147         nlapse++;
00148     }
00149     lapse_mean /= nlapse;
00150     lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00151
00152     /* Get histograms... */
00153     int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00154     if (idx >= 0 && idx < IDXMAX) {
00155         hist_max[idx]++;
00156         nhist_max++;
00157     }
00158
00159     idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00160     if (idx >= 0 && idx < IDXMAX) {
00161         hist_min[idx]++;
00162         nhist_min++;
00163     }
00164
00165     idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00166     if (idx >= 0 && idx < IDXMAX) {
00167         hist_mean[idx]++;
00168         nhist_mean++;
00169     }
00170
00171     idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00172     if (idx >= 0 && idx < IDXMAX) {
00173         hist_sig[idx]++;
00174         nhist_sig++;
00175     }
00176 }
00177 }
00178 }
00179
00180 /* Create output file... */
00181 LOG(1, "Write lapse rate data: %s", argv[2]);
00182 if (!(out = fopen(argv[2], "w")))
00183     ERRMSG("Cannot create file!");
00184
00185 /* Write header... */
00186 fprintf(out,
00187     "# $1 = mean altitude [km]\n"
00188     "# $2 = mean latitude [deg]\n"
00189     "# $3 = lapse rate [K/km]\n"
00190     "# $4 = counts of maxima per bin\n"
00191     "# $5 = total number of maxima\n"
00192     "# $6 = normalized frequency of maxima\n"
00193     "# $7 = counts of minima per bin\n"
00194     "# $8 = total number of minima\n"
00195     "# $9 = normalized frequency of minima\n"
00196     "# $10 = counts of means per bin\n"
00197     "# $11 = total number of means\n"
00198     "# $12 = normalized frequency of means\n"
00199     "# $13 = counts of sigmas per bin\n"
00200     "# $14 = total number of sigmas\n"
00201     "# $15 = normalized frequency of sigmas\n\n");
00202
00203 /* Write data... */
00204 double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00205 for (int idx = 0; idx < IDXMAX; idx++) {
00206     nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00207     nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00208     nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00209     nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00210 }
00211 for (int idx = 0; idx < IDXMAX; idx++)
00212     fprintf(out,

```

```
00213         "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
00214         z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00215         hist_max[idx], nhist_max,
00216         (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00217         nhist_min, (double) hist_min[idx] / (double) nmax_min,
00218         hist_mean[idx], nhist_mean,
00219         (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00220         nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00221
00222     /* Close file... */
00223     fclose(out);
00224
00225     /* Free... */
00226     free(clim);
00227     free(met);
00228
00229     return EXIT_SUCCESS;
00230 }
```



Generated by Doxygen

```

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     clim_t *clim;
00051
00052     met_t *met;
00053
00054     FILE *out;
00055
00056     static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00057         z_mean;
00058
00059     static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00060         nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <lapse.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00073     double lat0 =
00074         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00075     double lat1 =
00076         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00077     double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00078     double z1 =
00079         (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00080     int intpol =
00081         (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);
00082
00083     /* Read climatological data... */
00084     read_clim(&ctl, clim);
00085
00086     /* Loop over files... */
00087     for (int i = 3; i < argc; i++) {
00088
00089         /* Read meteorological data... */
00090         if (!read_met(argv[i], &ctl, clim, met))
00091             continue;
00092
00093         /* Get altitude and pressure profiles... */
00094         for (int iz = 0; iz < met->np; iz++)
00095             z[iz] = Z(met->p[iz]);
00096         for (int iz = 0; iz <= 250; iz++) {
00097             z2[iz] = 0.0 + 0.1 * iz;
00098             p2[iz] = P(z2[iz]);
00099         }
00100
00101         /* Loop over grid points... */
00102         for (int ix = 0; ix < met->nx; ix++)
00103             for (int iy = 0; iy < met->ny; iy++) {
00104
00105                 /* Check latitude range... */
00106                 if (met->lat[iy] < lat0 || met->lat[iy] > lat1)

```

```

00107         continue;
00108
00109     /* Interpolate temperature profile... */
00110     for (int iz = 0; iz < met->np; iz++)
00111         t[iz] = met->t[ix][iy][iz];
00112     if (intpol == 1)
00113         spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
00114     else
00115         for (int iz = 0; iz <= 250; iz++) {
00116             int idx = locate_irr(z, met->np, z2[iz]);
00117             t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
00118         }
00119
00120     /* Loop over vertical levels... */
00121     for (int iz = 0; iz <= 250; iz++) {
00122
00123         /* Check height range... */
00124         if (z2[iz] < z0 || z2[iz] > z1)
00125             continue;
00126
00127         /* Check surface pressure... */
00128         if (p2[iz] > met->ps[ix][iy])
00129             continue;
00130
00131         /* Get mean latitude and height... */
00132         lat_mean += met->lat[iy];
00133         z_mean += z2[iz];
00134         np++;
00135
00136         /* Get lapse rates within a vertical layer... */
00137         int nlapse = 0;
00138         double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00139             0, lapse_sig = 0;
00140         for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00141             lapse_max =
00142                 GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00143             lapse_min =
00144                 GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00145             lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00146             lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00147             nlapse++;
00148         }
00149         lapse_mean /= nlapse;
00150         lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00151
00152         /* Get histograms... */
00153         int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00154         if (idx >= 0 && idx < IDXMAX) {
00155             hist_max[idx]++;
00156             nhist_max++;
00157         }
00158
00159         idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00160         if (idx >= 0 && idx < IDXMAX) {
00161             hist_min[idx]++;
00162             nhist_min++;
00163         }
00164
00165         idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00166         if (idx >= 0 && idx < IDXMAX) {
00167             hist_mean[idx]++;
00168             nhist_mean++;
00169         }
00170
00171         idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00172         if (idx >= 0 && idx < IDXMAX) {
00173             hist_sig[idx]++;
00174             nhist_sig++;
00175         }
00176     }
00177 }
00178 }
00179
00180 /* Create output file... */
00181 LOG(1, "Write lapse rate data: %s", argv[2]);
00182 if (!(out = fopen(argv[2], "w")))
00183     ERRMSG("Cannot create file!");
00184
00185 /* Write header... */
00186 fprintf(out,
00187     "# $1 = mean altitude [km]\n"
00188     "# $2 = mean latitude [deg]\n"
00189     "# $3 = lapse rate [K/km]\n"
00190     "# $4 = counts of maxima per bin\n"
00191     "# $5 = total number of maxima\n"
00192     "# $6 = normalized frequency of maxima\n"
00193     "# $7 = counts of minima per bin\n"

```



```

00194         "# $8 = total number of minima\n"
00195         "# $9 = normalized frequency of minima\n"
00196         "# $10 = counts of means per bin\n"
00197         "# $11 = total number of means\n"
00198         "# $12 = normalized frequency of means\n"
00199         "# $13 = counts of sigmas per bin\n"
00200         "# $14 = total number of sigmas\n"
00201         "# $15 = normalized frequency of sigmas\n\n");
00202
00203     /* Write data... */
00204     double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00205     for (int idx = 0; idx < IDXMAX; idx++) {
00206         nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00207         nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00208         nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00209         nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00210     }
00211     for (int idx = 0; idx < IDXMAX; idx++)
00212         fprintf(out,
00213             "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
00214             z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00215             hist_max[idx], nhist_max,
00216             (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00217             nhist_min, (double) hist_min[idx] / (double) nmax_min,
00218             hist_mean[idx], nhist_mean,
00219             (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00220             nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00221
00222     /* Close file... */
00223     fclose(out);
00224
00225     /* Free... */
00226     free(clim);
00227     free(met);
00228
00229     return EXIT_SUCCESS;
00230 }

```

## 5.27 met\_map.c File Reference

Extract map from meteorological data.

```
#include "libtrac.h"
```

### Macros

- `#define NX 1441`  
*Maximum number of longitudes.*
- `#define NY 721`  
*Maximum number of latitudes.*

### Functions

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

#### 5.27.1 Detailed Description

Extract map from meteorological data.

Definition in file [met\\_map.c](#).

## 5.27.2 Macro Definition Documentation

### 5.27.2.1 NX #define NX 1441

Maximum number of longitudes.

Definition at line 32 of file [met\\_map.c](#).

### 5.27.2.2 NY #define NY 721

Maximum number of latitudes.

Definition at line 35 of file [met\\_map.c](#).

## 5.27.3 Function Documentation

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

Definition at line 41 of file [met\\_map.c](#).

```

00043     {
00044
00045     ctl_t ctl;
00046
00047     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
00054         zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
00055         ptm[NX][NY], t, pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY],
00056         w, wm[NX][NY], h2o, h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00057         hno3m[NX][NY], ohm[NX][NY], h2o2m[NX][NY], tdewm[NX][NY], ticem[NX][NY],
00058         tnatm[NX][NY], lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
00059         pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY], pct, pctm[NX][NY], pcb,
00060         pcbm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
00061         pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
00062         rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
00063         lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00064
00065     static int i, ix, iy, np[NX][NY], npc[NX][NY], npt[NX][NY], nx, ny, ci[3];
00066
00067     /* Allocate... */
00068     ALLOC(clim, clim_t, 1);
00069     ALLOC(met, met_t, 1);
00070
00071     /* Check arguments... */
00072     if (argc < 4)
00073         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00074
00075     /* Read control parameters... */
00076     read_ctl(argv[1], argc, argv, &ctl);
00077     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00078     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00079     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00080     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00081     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);

```

```

00082 lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00083 dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00084 theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00085
00086 /* Read climatological data... */
00087 read_clim(&ctl, clim);
00088
00089 /* Loop over files... */
00090 for (i = 3; i < argc; i++) {
00091
00092     /* Read meteorological data... */
00093     if (!read_met(argv[i], &ctl, clim, met))
00094         continue;
00095
00096     /* Set horizontal grid... */
00097     if (dlon <= 0)
00098         dlon = fabs(met->lon[1] - met->lon[0]);
00099     if (dlat <= 0)
00100         dlat = fabs(met->lat[1] - met->lat[0]);
00101     if (lon0 < -360 && lon1 > 360) {
00102         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00103         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00104     }
00105     nx = ny = 0;
00106     for (lon = lon0; lon <= lon1; lon += dlon) {
00107         lons[nx] = lon;
00108         if (++nx > NX)
00109             ERRMSG("Too many longitudes!");
00110     }
00111     if (lat0 < -90 && lat1 > 90) {
00112         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00113         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00114     }
00115     for (lat = lat0; lat <= lat1; lat += dlat) {
00116         lats[ny] = lat;
00117         if (++ny > NY)
00118             ERRMSG("Too many latitudes!");
00119     }
00120
00121     /* Average... */
00122     for (ix = 0; ix < nx; ix++)
00123         for (iy = 0; iy < ny; iy++) {
00124
00125             /* Find pressure level for given theta level... */
00126             if (theta > 0) {
00127                 ptop = met->p[met->np - 1];
00128                 pbot = met->p[0];
00129                 do {
00130                     p0 = 0.5 * (ptop + pbot);
00131                     intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00132                                         &t0, ci, cw, 1);
00133                     if (THETA(p0, t0) > theta)
00134                         ptop = p0;
00135                     else
00136                         pbot = p0;
00137                 } while (fabs(ptop - pbot) > 1e-5);
00138             }
00139
00140             /* Interpolate meteo data... */
00141             INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00142
00143             /* Averaging... */
00144             timem[ix][iy] += met->time;
00145             zm[ix][iy] += z;
00146             pm[ix][iy] += p0;
00147             tm[ix][iy] += t;
00148             um[ix][iy] += u;
00149             vm[ix][iy] += v;
00150             wm[ix][iy] += w;
00151             pvm[ix][iy] += pv;
00152             h2om[ix][iy] += h2o;
00153             o3m[ix][iy] += o3;
00154             lwcm[ix][iy] += lwc;
00155             iwcm[ix][iy] += iwc;
00156             psm[ix][iy] += ps;
00157             tsm[ix][iy] += ts;
00158             zsm[ix][iy] += zs;
00159             usm[ix][iy] += us;
00160             vsm[ix][iy] += vs;
00161             pblm[ix][iy] += pbl;
00162             pctm[ix][iy] += pct;
00163             pcbm[ix][iy] += pcb;
00164             clm[ix][iy] += cl;
00165             if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00166                 && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00167                 plclm[ix][iy] += plcl;
00168                 plfcm[ix][iy] += plfc;

```

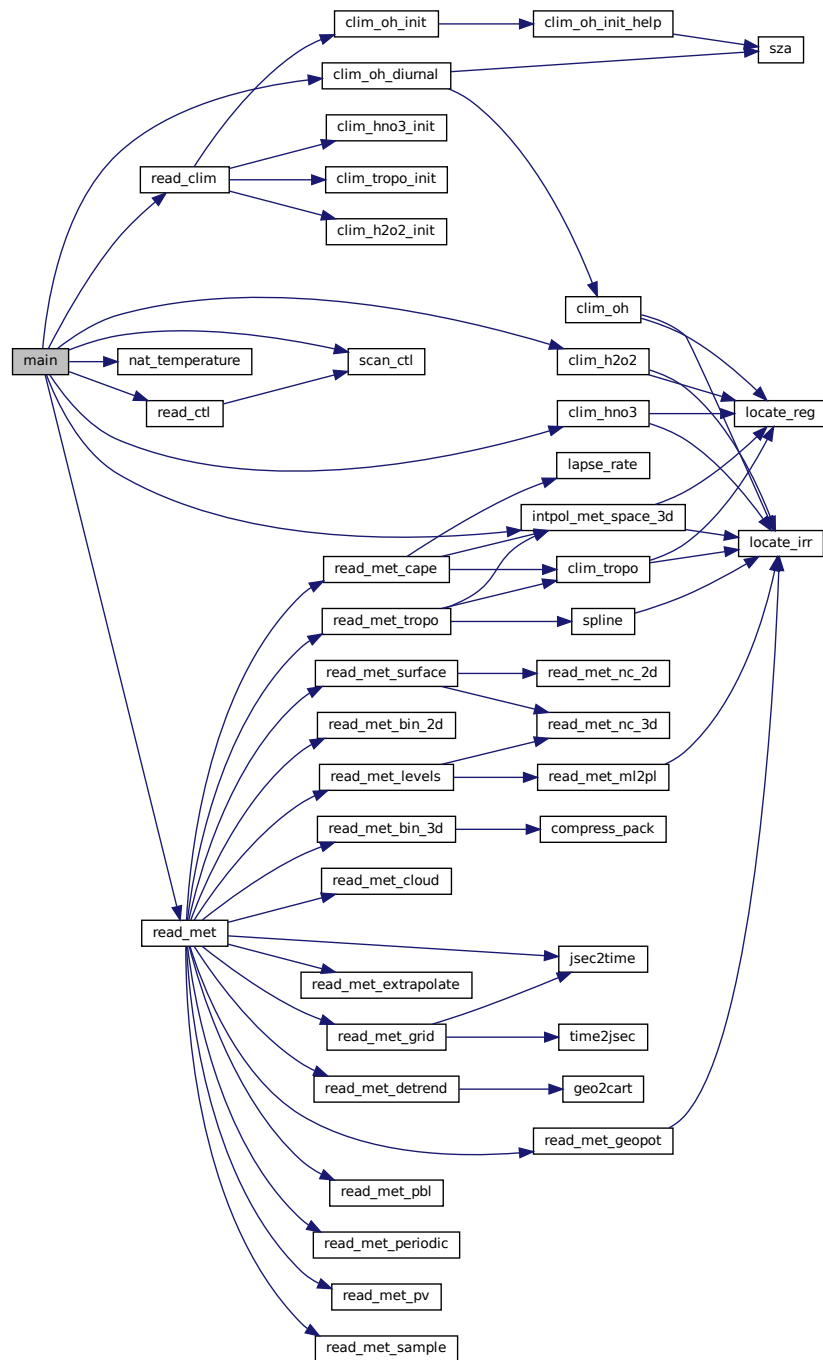
```

00169         pelm[ix][iy] += pel;
00170         capem[ix][iy] += cape;
00171         cinm[ix][iy] += cin;
00172         npc[ix][iy]++;
00173     }
00174     if (gsl_finite(pt)) {
00175         ptm[ix][iy] += pt;
00176         ztm[ix][iy] += zt;
00177         ttm[ix][iy] += tt;
00178         h2otm[ix][iy] += h2ot;
00179         npt[ix][iy]++;
00180     }
00181     hno3m[ix][iy] += clim_hno3(clim, met->time, lats[iy], p0);
00182     tnatm[ix][iy] +=
00183         nat_temperature(p0, h2o, clim_hno3(clim, met->time, lats[iy], p0));
00184     ohm[ix][iy] +=
00185         clim_oh_diurnal(&ctl, clim, met->time, p0, lons[ix], lats[iy]);
00186     h2o2m[ix][iy] += clim_h2o2(clim, met->time, lats[iy], p0);
00187     rhm[ix][iy] += RH(p0, t, h2o);
00188     rhicem[ix][iy] += RHICE(p0, t, h2o);
00189     tdewm[ix][iy] += TDEW(p0, h2o);
00190     ticem[ix][iy] += TICE(p0, h2o);
00191     np[ix][iy]++;
00192 }
00193 }
00194
00195 /* Create output file... */
00196 LOG(1, "Write meteorological data file: %s", argv[2]);
00197 if (!out = fopen(argv[2], "w"))
00198     ERRMSG("Cannot create file!");
00199
00200 /* Write header... */
00201 fprintf(out,
00202     "# $1 = time [s]\n"
00203     "# $2 = altitude [km]\n"
00204     "# $3 = longitude [deg]\n"
00205     "# $4 = latitude [deg]\n"
00206     "# $5 = pressure [hPa]\n"
00207     "# $6 = temperature [K]\n"
00208     "# $7 = zonal wind [m/s]\n"
00209     "# $8 = meridional wind [m/s]\n"
00210     "# $9 = vertical velocity [hPa/s]\n"
00211     "# $10 = H2O volume mixing ratio [ppv]\n");
00212 fprintf(out,
00213     "# $11 = O3 volume mixing ratio [ppv]\n"
00214     "# $12 = geopotential height [km]\n"
00215     "# $13 = potential vorticity [PVU]\n"
00216     "# $14 = surface pressure [hPa]\n"
00217     "# $15 = surface temperature [K]\n"
00218     "# $16 = surface geopotential height [km]\n"
00219     "# $17 = surface zonal wind [m/s]\n"
00220     "# $18 = surface meridional wind [m/s]\n"
00221     "# $19 = tropopause pressure [hPa]\n"
00222     "# $20 = tropopause geopotential height [km]\n");
00223 fprintf(out,
00224     "# $21 = tropopause temperature [K]\n"
00225     "# $22 = tropopause water vapor [ppv]\n"
00226     "# $23 = cloud liquid water content [kg/kg]\n"
00227     "# $24 = cloud ice water content [kg/kg]\n"
00228     "# $25 = total column cloud water [kg/m^2]\n"
00229     "# $26 = cloud top pressure [hPa]\n"
00230     "# $27 = cloud bottom pressure [hPa]\n"
00231     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00232     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00233     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00234 fprintf(out,
00235     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00236     "# $32 = convective inhibition (CIN) [J/kg]\n"
00237     "# $33 = relative humidity over water [%]\n"
00238     "# $34 = relative humidity over ice [%]\n"
00239     "# $35 = dew point temperature [K]\n"
00240     "# $36 = frost point temperature [K]\n"
00241     "# $37 = NAT temperature [K]\n"
00242     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00243     "# $39 = OH concentration [molec/cm^3]\n"
00244     "# $40 = H2O2 concentration [molec/cm^3]\n");
00245 fprintf(out,
00246     "# $41 = boundary layer pressure [hPa]\n"
00247     "# $42 = number of data points\n"
00248     "# $43 = number of tropopause data points\n"
00249     "# $44 = number of CAPE data points\n");
00250
00251 /* Write data... */
00252 for (iy = 0; iy < ny; iy++) {
00253     fprintf(out, "\n");
00254     for (ix = 0; ix < nx; ix++)
00255         fprintf(out,

```

```
00256         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g"
00257         " %g %g %g %g %g %g %g %g %g %g %g %g %g %g"
00258         " %g %g %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
00259         timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00260         lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00261         tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00262         vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00263         h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00264         zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00265         psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00266         zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00267         vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
00268         ztm[ix][iy] / npt[ix][iy], ttm[ix][iy] / npt[ix][iy],
00269         h2otm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
00270         iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00271         pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
00272         plclm[ix][iy] / npc[ix][iy], plfcm[ix][iy] / npc[ix][iy],
00273         pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
00274         cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
00275         rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
00276         ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
00277         hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
00278         h2o2m[ix][iy] / np[ix][iy], pblm[ix][iy] / np[ix][iy],
00279         np[ix][iy], npt[ix][iy], npc[ix][iy]);
00280     }
00281
00282     /* Close file... */
00283     fclose(out);
00284
00285     /* Free... */
00286     free(clim);
00287     free(met);
00288
00289     return EXIT_SUCCESS;
00290 }
```

Here is the call graph for this function:



## 5.28 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) 2013-2022 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     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
00054         zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
00055         ptm[NX][NY], t, pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY],
00056         w, wm[NX][NY], h2o, h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00057         hno3m[NX][NY], ohm[NX][NY], h2o2m[NX][NY], tdewm[NX][NY], ticem[NX][NY],
00058         tnatm[NX][NY], lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
00059         pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY], pct, pctm[NX][NY], pcb,
00060         pcbm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
00061         pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
00062         rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
00063         lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00064
00065     static int i, ix, iy, np[NX][NY], npc[NX][NY], npt[NX][NY], nx, ny, ci[3];
00066
00067     /* Allocate... */
00068     ALLOC(clim, clim_t, 1);
00069     ALLOC(met, met_t, 1);
00070
00071     /* Check arguments... */
00072     if (argc < 4)
00073         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00074
00075     /* Read control parameters... */
00076     read_ctl(argv[1], argc, argv, &ctl);
00077     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00078     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00079     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00080     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00081     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00082     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00083     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00084     theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00085
00086     /* Read climatological data... */
00087     read_clim(&ctl, clim);
00088
00089     /* Loop over files... */
00090     for (i = 3; i < argc; i++) {
00091
00092         /* Read meteorological data... */
00093         if (!read_met(argv[i], &ctl, clim, met))
00094             continue;
00095
00096         /* Set horizontal grid... */
00097         if (dlon <= 0)
00098             dlon = fabs(met->lon[1] - met->lon[0]);
00099         if (dlat <= 0)
00100             dlat = fabs(met->lat[1] - met->lat[0]);
00101         if (lon0 < -360 && lon1 > 360) {
00102             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);

```

```

00103     lonl = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00104 }
00105 nx = ny = 0;
00106 for (lon = lon0; lon <= lonl; lon += dlon) {
00107     lons[nx] = lon;
00108     if (++nx > NX)
00109         ERRMSG("Too many longitudes!");
00110 }
00111 if (lat0 < -90 && latl > 90) {
00112     lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00113     latl = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00114 }
00115 for (lat = lat0; lat <= latl; lat += dlat) {
00116     lats[ny] = lat;
00117     if (++ny > NY)
00118         ERRMSG("Too many latitudes!");
00119 }
00120
00121 /* Average... */
00122 for (ix = 0; ix < nx; ix++)
00123     for (iy = 0; iy < ny; iy++) {
00124
00125         /* Find pressure level for given theta level... */
00126         if (theta > 0) {
00127             ptop = met->p[met->np - 1];
00128             pbot = met->p[0];
00129             do {
00130                 p0 = 0.5 * (ptop + pbot);
00131                 intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00132                                     &t0, ci, cw, 1);
00133                 if (THETA(p0, t0) > theta)
00134                     ptop = p0;
00135                 else
00136                     pbot = p0;
00137             } while (fabs(ptop - pbot) > 1e-5);
00138         }
00139
00140         /* Interpolate meteo data... */
00141         INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00142
00143         /* Averaging... */
00144         timem[ix][iy] += met->time;
00145         zm[ix][iy] += z;
00146         pm[ix][iy] += p0;
00147         tm[ix][iy] += t;
00148         um[ix][iy] += u;
00149         vm[ix][iy] += v;
00150         wm[ix][iy] += w;
00151         pvm[ix][iy] += pv;
00152         h2om[ix][iy] += h2o;
00153         o3m[ix][iy] += o3;
00154         lwcm[ix][iy] += lwc;
00155         iwcm[ix][iy] += iwc;
00156         psm[ix][iy] += ps;
00157         tsm[ix][iy] += ts;
00158         zsm[ix][iy] += zs;
00159         usm[ix][iy] += us;
00160         vsm[ix][iy] += vs;
00161         pblm[ix][iy] += pbl;
00162         pctm[ix][iy] += pct;
00163         pcbm[ix][iy] += pcb;
00164         clm[ix][iy] += cl;
00165         if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00166             && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00167             plclm[ix][iy] += plcl;
00168             plfcm[ix][iy] += plfc;
00169             pelm[ix][iy] += pel;
00170             capem[ix][iy] += cape;
00171             cinm[ix][iy] += cin;
00172             npc[ix][iy]++;
00173         }
00174         if (gsl_finite(pt)) {
00175             ptm[ix][iy] += pt;
00176             ztm[ix][iy] += zt;
00177             ttm[ix][iy] += tt;
00178             h2otm[ix][iy] += h2ot;
00179             npt[ix][iy]++;
00180         }
00181         hno3m[ix][iy] += clim_hno3(clim, met->time, lats[iy], p0);
00182         tnatm[ix][iy] +=
00183             nat_temperature(p0, h2o, clim_hno3(clim, met->time, lats[iy], p0));
00184         ohm[ix][iy] +=
00185             clim_oh_diurnal(&ctl, clim, met->time, p0, lons[ix], lats[iy]);
00186         h2o2m[ix][iy] += clim_h2o2(clim, met->time, lats[iy], p0);
00187         rhm[ix][iy] += RH(p0, t, h2o);
00188         rhicem[ix][iy] += RHICE(p0, t, h2o);
00189         tdewm[ix][iy] += TDEW(p0, h2o);

```



[illegible]

```
00277             hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
00278             h2o2m[ix][iy] / np[ix][iy], pblm[ix][iy] / np[ix][iy],
00279             np[ix][iy], npt[ix][iy], npc[ix][iy]);
00280     }
00281
00282     /* Close file... */
00283     fclose(out);
00284
00285     /* Free... */
00286     free(clim);
00287     free(met);
00288
00289     return EXIT_SUCCESS;
00290 }
```

## 5.29 met\_prof.c File Reference

Extract vertical profile from meteorological data.

```
#include "libtrac.h"
```

### Macros

- `#define NZ 1000`  
*Maximum number of altitudes.*

### Functions

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

#### 5.29.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file [met\\_prof.c](#).

#### 5.29.2 Macro Definition Documentation

##### 5.29.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line [32](#) of file [met\\_prof.c](#).

#### 5.29.3 Function Documentation

```

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

```

Definition at line 38 of file [met\\_prof.c](#).

```

00040     {
00041
00042     ctl_t ctl;
00043
00044     clim_t *clim;
00045
00046     met_t *met;
00047
00048     FILE *out;
00049
00050     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00051     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00052     wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00053     iwc, iwcm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00054     vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
00055     cl, clim[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
00056     cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
00057     pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
00058     tnatm[NZ], hno3m[NZ], ohm[NZ], h2o2m[NZ], cw[3];
00059
00060     static int i, iz, np[NZ], npc[NZ], npt[NZ], nz, ci[3];
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00073     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00074     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00075     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00076     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00077     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00078     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00079     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00080     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00081
00082     /* Read climatological data... */
00083     read_clim(&ctl, clim);
00084
00085     /* Loop over input files... */
00086     for (i = 3; i < argc; i++) {
00087
00088         /* Read meteorological data... */
00089         if (!read_met(argv[i], &ctl, clim, met))
00090             continue;
00091
00092         /* Set vertical grid... */
00093         if (z0 < 0)
00094             z0 = Z(met->p[0]);
00095         if (z1 < 0)
00096             z1 = Z(met->p[met->np - 1]);
00097         nz = 0;
00098         if (dz < 0) {
00099             for (iz = 0; iz < met->np; iz++)
00100                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00101                     plev[nz] = met->p[iz];
00102                     if (++nz > NZ)
00103                         ERRMSG("Too many pressure levels!");
00104                 }
00105             } else
00106                 for (z = z0; z <= z1; z += dz) {
00107                     plev[nz] = P(z);
00108                     if (++nz > NZ)
00109                         ERRMSG("Too many pressure levels!");
00110                 }
00111
00112         /* Set horizontal grid... */
00113         if (dlon <= 0)
00114             dlon = fabs(met->lon[1] - met->lon[0]);
00115         if (dlat <= 0)
00116             dlat = fabs(met->lat[1] - met->lat[0]);
00117
00118         /* Average... */
00119         for (iz = 0; iz < nz; iz++)

```

```

00120     for (lon = lon0; lon <= lon1; lon += dlon)
00121         for (lat = lat0; lat <= lat1; lat += dlat) {
00122
00123             /* Interpolate meteo data... */
00124             INTPOL_SPACE_ALL(plev[iz], lon, lat);
00125
00126             /* Averaging... */
00127             if (gsl_finite(t) && gsl_finite(u)
00128                 && gsl_finite(v) && gsl_finite(w)) {
00129                 timem[iz] += met->time;
00130                 lonm[iz] += lon;
00131                 latm[iz] += lat;
00132                 zm[iz] += z;
00133                 tm[iz] += t;
00134                 um[iz] += u;
00135                 vm[iz] += v;
00136                 wm[iz] += w;
00137                 pvm[iz] += pv;
00138                 h2om[iz] += h2o;
00139                 o3m[iz] += o3;
00140                 lwcm[iz] += lwc;
00141                 iwcm[iz] += iwc;
00142                 psm[iz] += ps;
00143                 tsm[iz] += ts;
00144                 zsm[iz] += zs;
00145                 usm[iz] += us;
00146                 vsm[iz] += vs;
00147                 pblm[iz] += pbl;
00148                 pctm[iz] += pct;
00149                 pcbm[iz] += pcb;
00150                 clm[iz] += cl;
00151                 if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00152                     && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00153                     plclm[iz] += plcl;
00154                     plfcm[iz] += plfc;
00155                     pelm[iz] += pel;
00156                     capem[iz] += cape;
00157                     cinm[iz] += cin;
00158                     npc[iz]++;
00159                 }
00160                 if (gsl_finite(pt)) {
00161                     ptm[iz] += pt;
00162                     ztm[iz] += zt;
00163                     ttm[iz] += tt;
00164                     h2otm[iz] += h2ot;
00165                     npt[iz]++;
00166                 }
00167                 rhm[iz] += RH(plev[iz], t, h2o);
00168                 rhicem[iz] += RHICE(plev[iz], t, h2o);
00169                 tdewm[iz] += TDEW(plev[iz], h2o);
00170                 ticem[iz] += TICE(plev[iz], h2o);
00171                 hno3m[iz] += clim_hno3(clim, met->time, lat, plev[iz]);
00172                 tnatm[iz] +=
00173                     nat_temperature(plev[iz], h2o,
00174                                     clim_hno3(clim, met->time, lat, plev[iz]));
00175                 ohm[iz] +=
00176                     clim_oh_diurnal(&ctl, clim, met->time, plev[iz], lon, lat);
00177                 h2o2m[iz] += clim_h2o2(clim, met->time, lat, plev[iz]);
00178                 np[iz]++;
00179             }
00180         }
00181     }
00182
00183     /* Create output file... */
00184     LOG(1, "Write meteorological data file: %s", argv[2]);
00185     if (!(out = fopen(argv[2], "w")))
00186         ERRMSG("Cannot create file!");
00187
00188     /* Write header... */
00189     fprintf(out,
00190         "# $1 = time [s]\n"
00191         "# $2 = altitude [km]\n"
00192         "# $3 = longitude [deg]\n"
00193         "# $4 = latitude [deg]\n"
00194         "# $5 = pressure [hPa]\n"
00195         "# $6 = temperature [K]\n"
00196         "# $7 = zonal wind [m/s]\n"
00197         "# $8 = meridional wind [m/s]\n"
00198         "# $9 = vertical velocity [hPa/s]\n"
00199         "# $10 = H2O volume mixing ratio [ppv]\n");
00200     fprintf(out,
00201         "# $11 = O3 volume mixing ratio [ppv]\n"
00202         "# $12 = geopotential height [km]\n"
00203         "# $13 = potential vorticity [PVU]\n"
00204         "# $14 = surface pressure [hPa]\n"
00205         "# $15 = surface temperature [K]\n"
00206         "# $16 = surface geopotential height [km]\n");

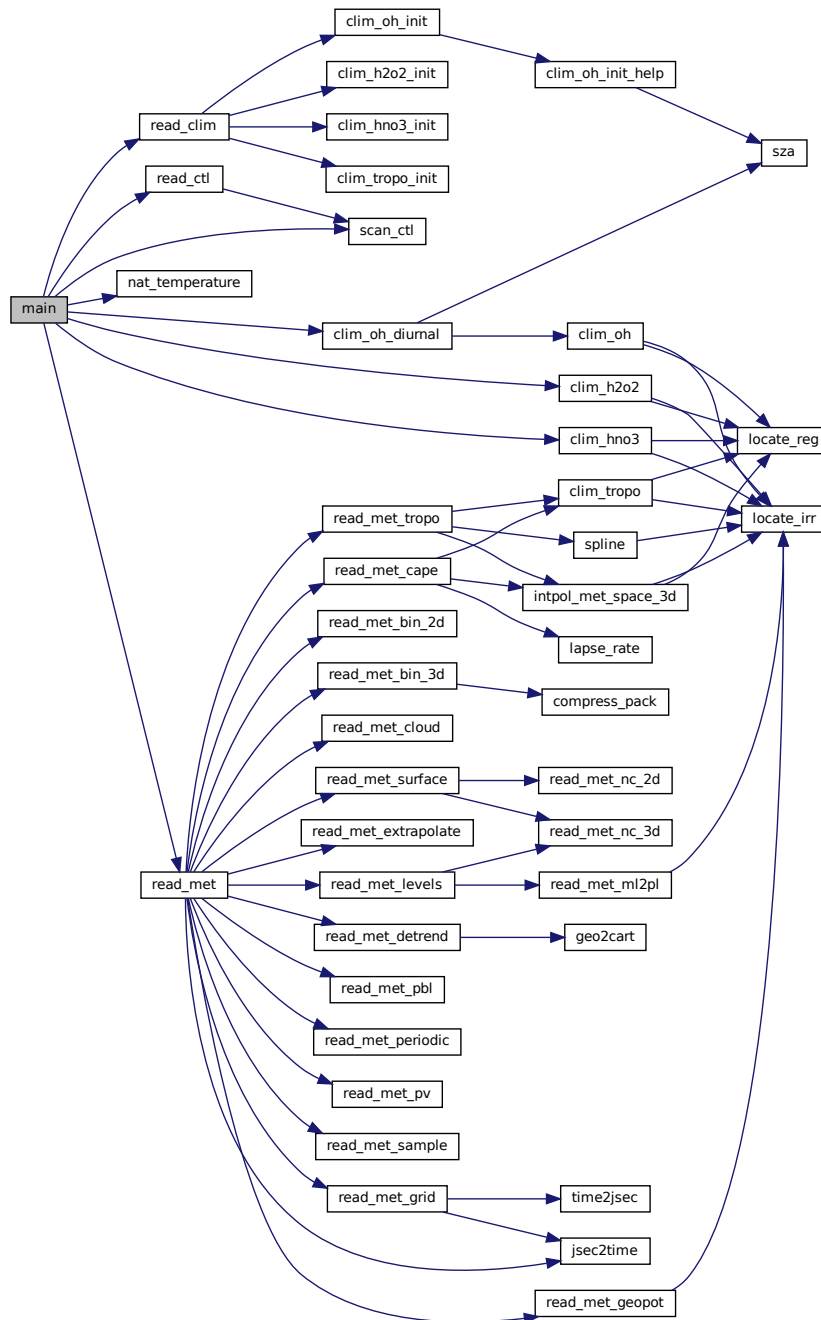
```

```

00207         "# $17 = surface zonal wind [m/s]\n"
00208         "# $18 = surface meridional wind [m/s]\n"
00209         "# $19 = tropopause pressure [hPa]\n"
00210         "# $20 = tropopause geopotential height [km]\n");
00211     fprintf(out,
00212         "# $21 = tropopause temperature [K]\n"
00213         "# $22 = tropopause water vapor [ppv]\n"
00214         "# $23 = cloud liquid water content [kg/kg]\n"
00215         "# $24 = cloud ice water content [kg/kg]\n"
00216         "# $25 = total column cloud water [kg/m^2]\n"
00217         "# $26 = cloud top pressure [hPa]\n"
00218         "# $27 = cloud bottom pressure [hPa]\n"
00219         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00220         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00221         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00222     fprintf(out,
00223         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00224         "# $32 = convective inhibition (CIN) [J/kg]\n"
00225         "# $33 = relative humidity over water [%]\n"
00226         "# $34 = relative humidity over ice [%]\n"
00227         "# $35 = dew point temperature [K]\n"
00228         "# $36 = frost point temperature [K]\n"
00229         "# $37 = NAT temperature [K]\n"
00230         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00231         "# $39 = OH concentration [molec/cm^3]\n"
00232         "# $40 = H2O2 concentration [molec/cm^3]\n");
00233     fprintf(out,
00234         "# $41 = boundary layer pressure [hPa]\n"
00235         "# $42 = number of data points\n"
00236         "# $43 = number of tropopause data points\n"
00237         "# $44 = number of CAPE data points\n\n");
00238
00239     /* Write data... */
00240     for (iz = 0; iz < nz; iz++)
00241         fprintf(out,
00242             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n"
00243             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00244             " %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
00245             timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00246             latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00247             vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00248             o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00249             psm[iz] / np[iz], tsm[iz] / np[iz], zsm[iz] / np[iz],
00250             usm[iz] / np[iz], vsm[iz] / np[iz], ptm[iz] / npt[iz],
00251             ztm[iz] / npt[iz], ttm[iz] / npt[iz], h2otm[iz] / npt[iz],
00252             lwcm[iz] / np[iz], iwcm[iz] / np[iz], clm[iz] / np[iz],
00253             pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
00254             plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
00255             cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
00256             tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
00257             hno3m[iz] / np[iz], ohm[iz] / np[iz], h2o2m[iz] / np[iz],
00258             pblm[iz] / np[iz], np[iz], npt[iz], npc[iz]);
00259
00260     /* Close file... */
00261     fclose(out);
00262
00263     /* Free... */
00264     free(clim);
00265     free(met);
00266
00267     return EXIT_SUCCESS;
00268 }

```

Here is the call graph for this function:



### 5.30 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) 2013-2022 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  /* -----
00028   Dimensions...
00029   ----- */
00030
00032  #define NZ 1000
00033
00034  /* -----
00035   Main...
00036   ----- */
00037
00038  int main(
00039      int argc,
00040      char *argv[]) {
00041
00042      ctl_t ctl;
00043
00044      clim_t *clim;
00045
00046      met_t *met;
00047
00048      FILE *out;
00049
00050      static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00051          lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00052          wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00053          iwc, iwcm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00054          vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
00055          cl, clm[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
00056          cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
00057          pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
00058          tnatm[NZ], hno3m[NZ], ohm[NZ], h2o2m[NZ], cw[3];
00059
00060      static int i, iz, np[NZ], npc[NZ], npt[NZ], nz, ci[3];
00061
00062      /* Allocate... */
00063      ALLOC(clim, clim_t, 1);
00064      ALLOC(met, met_t, 1);
00065
00066      /* Check arguments... */
00067      if (argc < 4)
00068          ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00069
00070      /* Read control parameters... */
00071      read_ctl(argv[1], argc, argv, &ctl);
00072      z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00073      z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00074      dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00075      lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00076      lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00077      dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00078      lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00079      lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00080      dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00081
00082      /* Read climatological data... */
00083      read_clim(&ctl, clim);
00084
00085      /* Loop over input files... */
00086      for (i = 3; i < argc; i++) {
00087
00088          /* Read meteorological data... */
00089          if (!read_met(argv[i], &ctl, clim, met))
00090              continue;
00091
00092          /* Set vertical grid... */
00093          if (z0 < 0)
00094              z0 = Z(met->p[0]);
00095          if (z1 < 0)
00096              z1 = Z(met->p[met->np - 1]);
00097          nz = 0;
00098          if (dz < 0) {
00099              for (iz = 0; iz < met->np; iz++)
00100                  if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00101                      plev[nz] = met->p[iz];
00102                      if (++nz > NZ)
00103                          ERRMSG("Too many pressure levels!");
00103

```

```

00104     }
00105 } else
00106     for (z = z0; z <= z1; z += dz) {
00107     plev[nz] = P(z);
00108     if ((++nz) > NZ)
00109         ERRMSG("Too many pressure levels!");
00110     }
00111
00112 /* Set horizontal grid... */
00113 if (dlon <= 0)
00114     dlon = fabs(met->lon[1] - met->lon[0]);
00115 if (dlat <= 0)
00116     dlat = fabs(met->lat[1] - met->lat[0]);
00117
00118 /* Average... */
00119 for (iz = 0; iz < nz; iz++)
00120     for (lon = lon0; lon <= lon1; lon += dlon)
00121         for (lat = lat0; lat <= lat1; lat += dlat) {
00122
00123             /* Interpolate meteo data... */
00124             INTPOL_SPACE_ALL(plev[iz], lon, lat);
00125
00126             /* Averaging... */
00127             if (gsl_finite(t) && gsl_finite(u)
00128                 && gsl_finite(v) && gsl_finite(w)) {
00129                 timem[iz] += met->time;
00130                 lonm[iz] += lon;
00131                 latm[iz] += lat;
00132                 zm[iz] += z;
00133                 tm[iz] += t;
00134                 um[iz] += u;
00135                 vm[iz] += v;
00136                 wm[iz] += w;
00137                 pvm[iz] += pv;
00138                 h2om[iz] += h2o;
00139                 o3m[iz] += o3;
00140                 lwcm[iz] += lwc;
00141                 iwcm[iz] += iwc;
00142                 psm[iz] += ps;
00143                 tsm[iz] += ts;
00144                 zsm[iz] += zs;
00145                 usm[iz] += us;
00146                 vsm[iz] += vs;
00147                 pblm[iz] += pbl;
00148                 pctm[iz] += pct;
00149                 pcbm[iz] += pcb;
00150                 clm[iz] += cl;
00151                 if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00152                     && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00153                     plclm[iz] += plcl;
00154                     plfcm[iz] += plfc;
00155                     pelm[iz] += pel;
00156                     capem[iz] += cape;
00157                     cinm[iz] += cin;
00158                     npc[iz]++;
00159                 }
00160                 if (gsl_finite(pt)) {
00161                     ptm[iz] += pt;
00162                     ztm[iz] += zt;
00163                     ttm[iz] += tt;
00164                     h2otm[iz] += h2ot;
00165                     npt[iz]++;
00166                 }
00167                 rhm[iz] += RH(plev[iz], t, h2o);
00168                 rhicem[iz] += RHICE(plev[iz], t, h2o);
00169                 tdewm[iz] += TDEW(plev[iz], h2o);
00170                 ticem[iz] += TICE(plev[iz], h2o);
00171                 hno3m[iz] += clim_hno3(clim, met->time, lat, plev[iz]);
00172                 tnatm[iz] +=
00173                     nat_temperature(plev[iz], h2o,
00174                                     clim_hno3(clim, met->time, lat, plev[iz]));
00175                 ohm[iz] +=
00176                     clim_oh_diurnal(&ctl, clim, met->time, plev[iz], lon, lat);
00177                 h2o2m[iz] += clim_h2o2(clim, met->time, lat, plev[iz]);
00178                 np[iz]++;
00179             }
00180         }
00181 }
00182
00183 /* Create output file... */
00184 LOG(1, "Write meteorological data file: %s", argv[2]);
00185 if (!(out = fopen(argv[2], "w")))
00186     ERRMSG("Cannot create file!");
00187
00188 /* Write header... */
00189 fprintf(out,
00190     "# $1 = time [s]\n"

```



[illegible]

### 5.31 met sample.c File Reference

Sample meteorological data at given geolocations.

```
#include "libtrac.h"
```

## Functions

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

### 5.31.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file [met\\_sample.c](#).

### 5.31.2 Function Documentation

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

Definition at line 31 of file [met\\_sample.c](#).

```
00033     {
00034
00035     ctl_t ctl;
00036
00037     clim_t *clim;
00038
00039     atm_t *atm;
00040
00041     met_t *met0, *met1;
00042
00043     FILE *out;
00044
00045     double h2o, h2ot, o3, lwc, iwc, p0, p1, ps, ts, zs, us, vs, pbl, pt,
00046            pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00047            zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00048
00049     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00050
00051     /* Check arguments... */
00052     if (argc < 3)
00053         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00054
00055     /* Allocate... */
00056     ALLOC(clim, clim_t, 1);
00057     ALLOC(atm, atm_t, 1);
00058     ALLOC(met0, met_t, 1);
00059     ALLOC(met1, met_t, 1);
00060
00061     /* Read control parameters... */
00062     read_ctl(argv[1], argc, argv, &ctl);
00063     geopot =
00064         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00065     grid_time =
00066         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00067     grid_z =
00068         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00069     grid_lon =
00070         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00071     grid_lat =
00072         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00073
00074     /* Read climatological data... */
00075     read_clim(&ctl, clim);
00076
00077     /* Read atmospheric data... */
00078     if (!read_atm(argv[3], &ctl, atm))
```

```

00079     ERRMSG("Cannot open file!");
00080
00081     /* Create output file... */
00082     LOG(1, "Write meteorological data file: %s", argv[2]);
00083     if (!out = fopen(argv[2], "w"))
00084         ERRMSG("Cannot create file!");
00085
00086     /* Write header... */
00087     fprintf(out,
00088         "# $1 = time [s]\n"
00089         "# $2 = altitude [km]\n"
00090         "# $3 = longitude [deg]\n"
00091         "# $4 = latitude [deg]\n"
00092         "# $5 = pressure [hPa]\n"
00093         "# $6 = temperature [K]\n"
00094         "# $7 = zonal wind [m/s]\n"
00095         "# $8 = meridional wind [m/s]\n"
00096         "# $9 = vertical velocity [hPa/s]\n"
00097         "# $10 = H2O volume mixing ratio [ppv]\n");
00098     fprintf(out,
00099         "# $11 = O3 volume mixing ratio [ppv]\n"
00100         "# $12 = geopotential height [km]\n"
00101         "# $13 = potential vorticity [PVU]\n"
00102         "# $14 = surface pressure [hPa]\n"
00103         "# $15 = surface temperature [K]\n"
00104         "# $16 = surface geopotential height [km]\n"
00105         "# $17 = surface zonal wind [m/s]\n"
00106         "# $18 = surface meridional wind [m/s]\n"
00107         "# $19 = tropopause pressure [hPa]\n"
00108         "# $20 = tropopause geopotential height [km]\n");
00109     fprintf(out,
00110         "# $21 = tropopause temperature [K]\n"
00111         "# $22 = tropopause water vapor [ppv]\n"
00112         "# $23 = cloud liquid water content [kg/kg]\n"
00113         "# $24 = cloud ice water content [kg/kg]\n"
00114         "# $25 = total column cloud water [kg/m^2]\n"
00115         "# $26 = cloud top pressure [hPa]\n"
00116         "# $27 = cloud bottom pressure [hPa]\n"
00117         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00118         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00119         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00120     fprintf(out,
00121         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00122         "# $32 = convective inhibition (CIN) [J/kg]\n"
00123         "# $33 = relative humidity over water [%]\n"
00124         "# $34 = relative humidity over ice [%]\n"
00125         "# $35 = dew point temperature [K]\n"
00126         "# $36 = frost point temperature [K]\n"
00127         "# $37 = NAT temperature [K]\n"
00128         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00129         "# $39 = OH concentration [molec/cm^3]\n"
00130         "# $40 = H2O2 concentration [molec/cm^3]\n"
00131         "# $41 = boundary layer pressure [hPa]\n");
00132
00133     /* Loop over air parcels... */
00134     for (ip = 0; ip < atm->np; ip++) {
00135
00136         /* Get meteorological data... */
00137         get_met(&ctl, clim, atm->time[ip], &met0, &met1);
00138
00139         /* Set reference pressure for interpolation... */
00140         double pref = atm->p[ip];
00141         if (geopot) {
00142             zref = Z(pref);
00143             p0 = met0->p[0];
00144             p1 = met0->p[met0->np - 1];
00145             for (it = 0; it < 24; it++) {
00146                 pref = 0.5 * (p0 + p1);
00147                 intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
00148                     atm->lon[ip], atm->lat[ip], &zm, ci, cw, l);
00149                 if (zref > zm || !gsl_finite(zm))
00150                     p0 = pref;
00151                 else
00152                     p1 = pref;
00153             }
00154             pref = 0.5 * (p0 + p1);
00155         }
00156
00157         /* Interpolate meteo data... */
00158         INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
00159
00160         /* Make blank lines... */
00161         if (ip == 0 || (grid_time && atm->time[ip] != time_old)
00162             || (grid_z && atm->p[ip] != p_old)
00163             || (grid_lon && atm->lon[ip] != lon_old)
00164             || (grid_lat && atm->lat[ip] != lat_old))
00165             fprintf(out, "\n");

```

```

00166     time_old = atm->time[ip];
00167     p_old = atm->p[ip];
00168     lon_old = atm->lon[ip];
00169     lat_old = atm->lat[ip];
00170
00171     /* Write data... */
00172     fprintf(out,
00173         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00174         " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00175         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00176         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
00177         pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
00178         cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
00179         TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
00180         nat_temperature(atm->p[ip], h2o,
00181             clim_hno3(clim, atm->time[ip], atm->lat[ip],
00182                 atm->p[ip])), clim_hno3(clim,
00183                 atm->time[ip],
00184                 atm->lat[ip],
00185                 atm->p[ip]),
00186         clim_oh_diurnal(&ctl, clim, atm->time[ip], atm->p[ip],
00187             atm->lon[ip], atm->lat[ip]),
00188         clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
00189 }
00190
00191 /* Close file... */
00192 fclose(out);
00193
00194 /* Free... */
00195 free(clim);
00196 free(atm);
00197 free(met0);
00198 free(met1);
00199
00200 return EXIT_SUCCESS;
00201 }

```



```

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     clim_t *clim;
00038
00039     atm_t *atm;
00040
00041     met_t *met0, *met1;
00042
00043     FILE *out;
00044
00045     double h2o, h2ot, o3, lwc, iwc, p0, pl, ps, ts, zs, us, vs, pbl, pt,
00046         pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00047         zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00048
00049     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00050
00051     /* Check arguments... */
00052     if (argc < 3)
00053         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00054
00055     /* Allocate... */
00056     ALLOC(clim, clim_t, 1);
00057     ALLOC(atm, atm_t, 1);
00058     ALLOC(met0, met_t, 1);
00059     ALLOC(met1, met_t, 1);
00060
00061     /* Read control parameters... */
00062     read_ctl(argv[1], argc, argv, &ctl);
00063     geopot =
00064         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00065     grid_time =
00066         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00067     grid_z =
00068         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00069     grid_lon =
00070         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00071     grid_lat =
00072         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00073
00074     /* Read climatological data... */
00075     read_clim(&ctl, clim);
00076
00077     /* Read atmospheric data... */
00078     if (!read_atm(argv[3], &ctl, atm))
00079         ERRMSG("Cannot open file!");
00080
00081     /* Create output file... */
00082     LOG(1, "Write meteorological data file: %s", argv[2]);
00083     if (!(out = fopen(argv[2], "w")))
00084         ERRMSG("Cannot create file!");
00085
00086     /* Write header... */
00087     fprintf(out,
00088         "# $1 = time [s]\n"
00089         "# $2 = altitude [km]\n"
00090         "# $3 = longitude [deg]\n"
00091         "# $4 = latitude [deg]\n"
00092         "# $5 = pressure [hPa]\n"
00093         "# $6 = temperature [K]\n"
00094         "# $7 = zonal wind [m/s]\n"
00095         "# $8 = meridional wind [m/s]\n"
00096         "# $9 = vertical velocity [hPa/s]\n"
00097         "# $10 = H2O volume mixing ratio [ppv]\n");
00098     fprintf(out,
00099         "# $11 = O3 volume mixing ratio [ppv]\n"
00100         "# $12 = geopotential height [km]\n"
00101         "# $13 = potential vorticity [PVU]\n"
00102         "# $14 = surface pressure [hPa]\n"
00103         "# $15 = surface temperature [K]\n"
00104         "# $16 = surface geopotential height [km]\n"
00105         "# $17 = surface zonal wind [m/s]\n"
00106         "# $18 = surface meridional wind [m/s]\n"
00107         "# $19 = tropopause pressure [hPa]\n"
00108         "# $20 = tropopause geopotential height [km]\n");
00109     fprintf(out,
00110         "# $21 = tropopause temperature [K]\n"

```

```

001111     "# $22 = tropopause water vapor [ppv]\n"
001112     "# $23 = cloud liquid water content [kg/kg]\n"
001113     "# $24 = cloud ice water content [kg/kg]\n"
001114     "# $25 = total column cloud water [kg/m^2]\n"
001115     "# $26 = cloud top pressure [hPa]\n"
001116     "# $27 = cloud bottom pressure [hPa]\n"
001117     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
001118     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
001119     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
001120 fprintf(out,
001121     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
001122     "# $32 = convective inhibition (CIN) [J/kg]\n"
001123     "# $33 = relative humidity over water [%]\n"
001124     "# $34 = relative humidity over ice [%]\n"
001125     "# $35 = dew point temperature [K]\n"
001126     "# $36 = frost point temperature [K]\n"
001127     "# $37 = NAT temperature [K]\n"
001128     "# $38 = HNO3 volume mixing ratio [ppv]\n"
001129     "# $39 = OH concentration [molec/cm^3]\n"
001130     "# $40 = H2O2 concentration [molec/cm^3]\n"
001131     "# $41 = boundary layer pressure [hPa]\n");
001132
001133 /* Loop over air parcels... */
001134 for (ip = 0; ip < atm->np; ip++) {
001135
001136     /* Get meteorological data... */
001137     get_met(&ctl, clim, atm->time[ip], &met0, &met1);
001138
001139     /* Set reference pressure for interpolation... */
001140     double pref = atm->p[ip];
001141     if (geopot) {
001142         zref = Z(pref);
001143         p0 = met0->p[0];
001144         p1 = met0->p[met0->np - 1];
001145         for (it = 0; it < 24; it++) {
001146             pref = 0.5 * (p0 + p1);
001147             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
001148                               atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
001149             if (zref > zm || !gs1_finite(zm))
001150                 p0 = pref;
001151             else
001152                 p1 = pref;
001153         }
001154         pref = 0.5 * (p0 + p1);
001155     }
001156
001157     /* Interpolate meteo data... */
001158     INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
001159
001160     /* Make blank lines... */
001161     if (ip == 0 || (grid_time && atm->time[ip] != time_old)
001162         || (grid_z && atm->p[ip] != p_old)
001163         || (grid_lon && atm->lon[ip] != lon_old)
001164         || (grid_lat && atm->lat[ip] != lat_old))
001165         fprintf(out, "\n");
001166     time_old = atm->time[ip];
001167     p_old = atm->p[ip];
001168     lon_old = atm->lon[ip];
001169     lat_old = atm->lat[ip];
001170
001171     /* Write data... */
001172     fprintf(out,
001173         "%2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
001174         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
001175         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
001176         pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
001177         cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
001178         TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
001179         nat_temperature(atm->p[ip], h2o,
001180             clim_hno3(clim, atm->time[ip], atm->lat[ip],
001181                       atm->p[ip])), clim_hno3(clim,
001182                                               atm->time[ip],
001183                                               atm->lat[ip],
001184                                               atm->p[ip]),
001185         clim_oh_diurnal(&ctl, clim, atm->time[ip], atm->p[ip],
001186                       atm->lon[ip], atm->lat[ip]),
001187         clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
001188 }
001189
001190 /* Close file... */
001191 fclose(out);
001192
001193 /* Free... */
001194 free(clim);
001195 free(atm);
001196 free(met0);

```

```
00198     free(met1);
00199
00200     return EXIT_SUCCESS;
00201 }
```

## 5.33 met\_spec.c File Reference

Spectral analysis of meteorological data.

```
#include "libtrac.h"
```

### Macros

- `#define PMAX EX`  
*Maximum number of data points for spectral analysis.*

### Functions

- void [fft\\_help](#) (double \*fcReal, double \*fclmag, int n)
- int [main](#) (int argc, char \*argv[])

#### 5.33.1 Detailed Description

Spectral analysis of meteorological data.

Definition in file [met\\_spec.c](#).

#### 5.33.2 Macro Definition Documentation

##### 5.33.2.1 PMAX `#define PMAX EX`

Maximum number of data points for spectral analysis.

Definition at line [32](#) of file [met\\_spec.c](#).

#### 5.33.3 Function Documentation



**5.33.3.1 fft\_help()** void fft\_help (  
double \* fcReal,  
double \* fcImag,  
int n )

Definition at line 150 of file [met\\_spec.c](#).

```
00153     {
00154
00155     gsl_fft_complex_wavetable *wavetable;
00156     gsl_fft_complex_workspace *workspace;
00157
00158     double data[2 * PMAX];
00159
00160     int i;
00161
00162     /* Check size... */
00163     if (n > PMAX)
00164         ERRMSG("Too many data points!");
00165
00166     /* Allocate... */
00167     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00168     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00169
00170     /* Set data (real, complex)... */
00171     for (i = 0; i < n; i++) {
00172         data[2 * i] = fcReal[i];
00173         data[2 * i + 1] = fcImag[i];
00174     }
00175
00176     /* Calculate FFT... */
00177     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00178
00179     /* Copy data... */
00180     for (i = 0; i < n; i++) {
00181         fcReal[i] = data[2 * i];
00182         fcImag[i] = data[2 * i + 1];
00183     }
00184
00185     /* Free... */
00186     gsl_fft_complex_wavetable_free(wavetable);
00187     gsl_fft_complex_workspace_free(workspace);
00188 }
```

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

Definition at line 47 of file [met\\_spec.c](#).

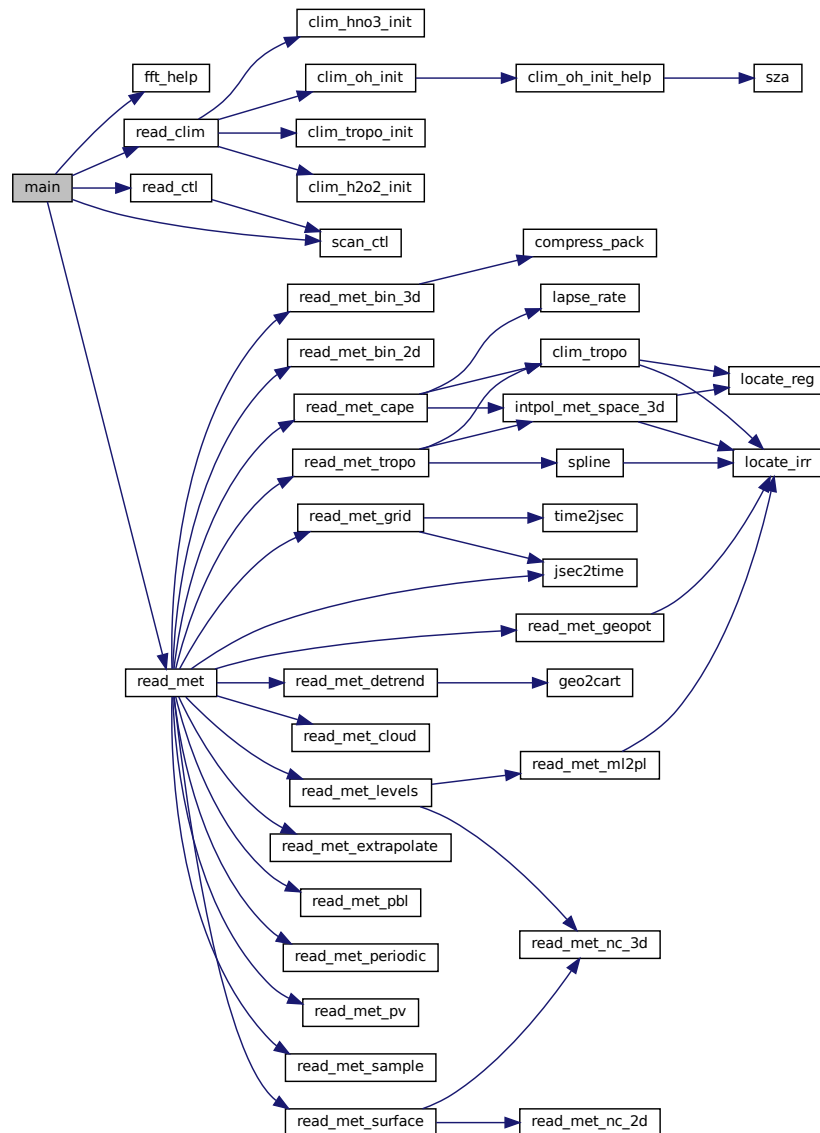
```
00049     {
00050
00051     ctl_t ctl;
00052
00053     clim_t *clim;
00054
00055     met_t *met;
00056
00057     FILE *out;
00058
00059     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00060         wavemax;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     wavemax =
00073         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00074
00075     /* Read climatological data... */
00076     read_clim(&ctl, clim);
00077 }
```

```

00078  /* Read meteorological data... */
00079  if (!read_met(argv[3], &ctl, clim, met))
00080      ERRMSG("Cannot read meteo data!");
00081
00082  /* Create output file... */
00083  LOG(1, "Write spectral data file: %s", argv[2]);
00084  if (!(out = fopen(argv[2], "w")))
00085      ERRMSG("Cannot create file!");
00086
00087  /* Write header... */
00088  fprintf(out,
00089      "# $1 = time [s]\n"
00090      "# $2 = altitude [km]\n"
00091      "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00092  for (int ix = 0; ix <= wavemax; ix++) {
00093      fprintf(out, "# $%d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
00094      fprintf(out, "# $%d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
00095      fprintf(out, "# $%d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
00096  }
00097
00098  /* Loop over pressure levels... */
00099  for (int ip = 0; ip < met->np; ip++) {
00100
00101      /* Write output... */
00102      fprintf(out, "\n");
00103
00104      /* Loop over latitudes... */
00105      for (int iy = 0; iy < met->ny; iy++) {
00106
00107          /* Copy data... */
00108          for (int ix = 0; ix < met->nx; ix++) {
00109              cutReal[ix] = met->t[ix][iy][ip];
00110              cutImag[ix] = 0.0;
00111          }
00112
00113          /* FFT... */
00114          fft_help(cutReal, cutImag, met->nx);
00115
00116          /*
00117           * Get wavelength, amplitude, and phase:
00118           *  $A(x) = A[0] + A[1] * \cos(2 \pi x / lx[1] + \phi[1]) + A[2] * \cos...$ 
00119           */
00120          for (int ix = 0; ix < met->nx; ix++) {
00121              lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00122                  / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00123              A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00124                  * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00125              phi[ix]
00126                  = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00127          }
00128
00129          /* Write data... */
00130          fprintf(out, "%.2f %g %g", met->time, Z(met->p[ip]), 0.0,
00131              met->lat[iy]);
00132          for (int ix = 0; ix <= wavemax; ix++)
00133              fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00134          fprintf(out, "\n");
00135      }
00136  }
00137
00138  /* Close file... */
00139  fclose(out);
00140
00141  /* Free... */
00142  free(clim);
00143  free(met);
00144
00145  return EXIT_SUCCESS;
00146 }

```

Here is the call graph for this function:



## 5.34 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     clim_t *clim;
00054
00055     met_t *met;
00056
00057     FILE *out;
00058
00059     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00060         wavemax;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     wavemax =
00073         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00074
00075     /* Read climatological data... */
00076     read_clim(&ctl, clim);
00077
00078     /* Read meteorological data... */
00079     if (!read_met(argv[3], &ctl, clim, met))
00080         ERRMSG("Cannot read meteo data!");
00081
00082     /* Create output file... */
00083     LOG(1, "Write spectral data file: %s", argv[2]);
00084     if (!(out = fopen(argv[2], "w")))
00085         ERRMSG("Cannot create file!");
00086
00087     /* Write header... */
00088     fprintf(out,
00089         "# $1 = time [s]\n"
00090         "# $2 = altitude [km]\n"
00091         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00092     for (int ix = 0; ix <= wavemax; ix++) {
00093         fprintf(out, "# $d = wavelength (PW$d) [km]\n", 5 + 3 * ix, ix);
00094         fprintf(out, "# $d = amplitude (PW$d) [K]\n", 6 + 3 * ix, ix);
00095         fprintf(out, "# $d = phase (PW$d) [deg]\n", 7 + 3 * ix, ix);
00096     }
00097
00098     /* Loop over pressure levels... */
00099     for (int ip = 0; ip < met->np; ip++) {
00100
00101         /* Write output... */
00102         fprintf(out, "\n");
00103
00104         /* Loop over latitudes... */
00105         for (int iy = 0; iy < met->ny; iy++) {
00106
00107             /* Copy data... */
00108             for (int ix = 0; ix < met->nx; ix++) {
00109                 cutReal[ix] = met->t[ix][iy][ip];
00110                 cutImag[ix] = 0.0;
00111             }
00112

```

```

00113      /* FFT... */
00114      fft_help(cutReal, cutImag, met->nx);
00115
00116      /*
00117       * Get wavelength, amplitude, and phase:
00118       *  $A(x) = A[0] + A[1] * \cos(2 \pi x / lx[1] + \phi[1]) + A[2] * \cos...$ 
00119       */
00120      for (int ix = 0; ix < met->nx; ix++) {
00121          lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00122              / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00123          A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00124              * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00125          phi[ix]
00126              = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00127      }
00128
00129      /* Write data... */
00130      fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
00131          met->lat[iy]);
00132      for (int ix = 0; ix <= wavemax; ix++)
00133          fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00134      fprintf(out, "\n");
00135  }
00136  }
00137
00138      /* Close file... */
00139      fclose(out);
00140
00141      /* Free... */
00142      free(clim);
00143      free(met);
00144
00145      return EXIT_SUCCESS;
00146  }
00147
00148  /*****
00149
00150 void fft_help(
00151     double *fcReal,
00152     double *fcImag,
00153     int n) {
00154
00155     gsl_fft_complex_wavetable *wavetable;
00156     gsl_fft_complex_workspace *workspace;
00157
00158     double data[2 * PMAX];
00159
00160     int i;
00161
00162     /* Check size... */
00163     if (n > PMAX)
00164         ERRMSG("Too many data points!");
00165
00166     /* Allocate... */
00167     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00168     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00169
00170     /* Set data (real, complex)... */
00171     for (i = 0; i < n; i++) {
00172         data[2 * i] = fcReal[i];
00173         data[2 * i + 1] = fcImag[i];
00174     }
00175
00176     /* Calculate FFT... */
00177     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00178
00179     /* Copy data... */
00180     for (i = 0; i < n; i++) {
00181         fcReal[i] = data[2 * i];
00182         fcImag[i] = data[2 * i + 1];
00183     }
00184
00185     /* Free... */
00186     gsl_fft_complex_wavetable_free(wavetable);
00187     gsl_fft_complex_workspace_free(workspace);
00188 }

```

## 5.35 met\_subgrid.c File Reference

Calculate standard deviations of horizontal wind and vertical velocity.

```
#include "libtrac.h"
```

## Functions

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

### 5.35.1 Detailed Description

Calculate standard deviations of horizontal wind and vertical velocity.

Definition in file [met\\_subgrid.c](#).

### 5.35.2 Function Documentation

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

Definition at line 31 of file [met\\_subgrid.c](#).

```
00033     {
00034
00035     ctl_t ctl;
00036
00037     clim_t *clim;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];
00044
00045     static float u[16], v[16], w[16];
00046
00047     static int i, ix, iy, iz, n[EP][EY];
00048
00049     /* Allocate... */
00050     ALLOC(clim, clim_t, 1);
00051     ALLOC(met0, met_t, 1);
00052     ALLOC(met1, met_t, 1);
00053
00054     /* Check arguments... */
00055     if (argc < 4 && argc % 2 != 0)
00056         ERRMSG
00057             ("Give parameters: <ctl> <subgrid.tab> <met0> <met1> [ <met0> <met1> ... ]");
00058
00059     /* Read control parameters... */
00060     read_ctl(argv[1], argc, argv, &ctl);
00061
00062     /* Read climatological data... */
00063     read_clim(&ctl, clim);
00064
00065     /* Loop over data files... */
00066     for (i = 3; i < argc - 1; i += 2) {
00067
00068         /* Read meteorological data... */
00069         if (!read_met(argv[i], &ctl, clim, met0))
00070             ERRMSG("Cannot open file!");
00071         if (!read_met(argv[i + 1], &ctl, clim, met1))
00072             ERRMSG("Cannot open file!");
00073
00074         /* Loop over grid boxes... */
00075         for (ix = 0; ix < met0->nx - 1; ix++)
00076             for (iy = 0; iy < met0->ny - 1; iy++)
00077                 for (iz = 0; iz < met0->np - 1; iz++) {
00078
00079                     /* Collect local wind data... */
00080                     u[0] = met0->u[ix][iy][iz];
00081                     u[1] = met0->u[ix + 1][iy][iz];
00082                     u[2] = met0->u[ix][iy + 1][iz];
00083                     u[3] = met0->u[ix + 1][iy + 1][iz];
00084                     u[4] = met0->u[ix][iy][iz + 1];
```

```

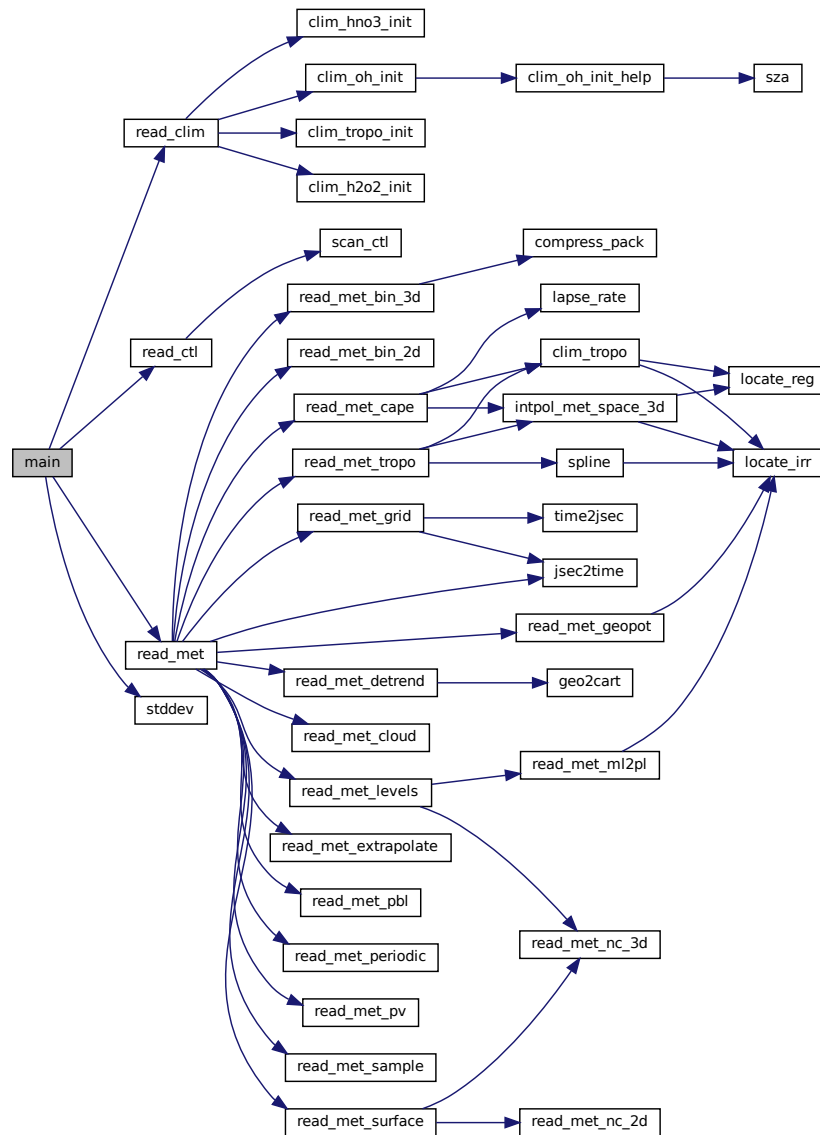
00085     u[5] = met0->u[ix + 1][iy][iz + 1];
00086     u[6] = met0->u[ix][iy + 1][iz + 1];
00087     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00088
00089     v[0] = met0->v[ix][iy][iz];
00090     v[1] = met0->v[ix + 1][iy][iz];
00091     v[2] = met0->v[ix][iy + 1][iz];
00092     v[3] = met0->v[ix + 1][iy + 1][iz];
00093     v[4] = met0->v[ix][iy][iz + 1];
00094     v[5] = met0->v[ix + 1][iy][iz + 1];
00095     v[6] = met0->v[ix][iy + 1][iz + 1];
00096     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00097
00098     w[0] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
00099     w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
00100     w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
00101     w[3] =
00102         (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
00103     w[4] =
00104         (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
00105     w[5] =
00106         (float) (1e3 *
00107             DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
00108     w[6] =
00109         (float) (1e3 *
00110             DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
00111     w[7] =
00112         (float) (1e3 *
00113             DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[iz + 1]));
00114
00115     /* Collect local wind data... */
00116     u[8] = met1->u[ix][iy][iz];
00117     u[9] = met1->u[ix + 1][iy][iz];
00118     u[10] = met1->u[ix][iy + 1][iz];
00119     u[11] = met1->u[ix + 1][iy + 1][iz];
00120     u[12] = met1->u[ix][iy][iz + 1];
00121     u[13] = met1->u[ix + 1][iy][iz + 1];
00122     u[14] = met1->u[ix][iy + 1][iz + 1];
00123     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00124
00125     v[8] = met1->v[ix][iy][iz];
00126     v[9] = met1->v[ix + 1][iy][iz];
00127     v[10] = met1->v[ix][iy + 1][iz];
00128     v[11] = met1->v[ix + 1][iy + 1][iz];
00129     v[12] = met1->v[ix][iy][iz + 1];
00130     v[13] = met1->v[ix + 1][iy][iz + 1];
00131     v[14] = met1->v[ix][iy + 1][iz + 1];
00132     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00133
00134     w[8] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
00135     w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
00136     w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
00137     w[11] =
00138         (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
00139     w[12] =
00140         (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
00141     w[13] =
00142         (float) (1e3 *
00143             DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
00144     w[14] =
00145         (float) (1e3 *
00146             DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
00147     w[15] =
00148         (float) (1e3 *
00149             DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));
00150
00151     /* Get standard deviations of local wind data... */
00152     usig[iz][iy] += stddev(u, 16);
00153     vsig[iz][iy] += stddev(v, 16);
00154     wsig[iz][iy] += stddev(w, 16);
00155     n[iz][iy]++;
00156
00157     /* Check surface pressure... */
00158     if (met0->p[iz] > met0->ps[ix][iy]
00159         || met1->p[iz] > met1->ps[ix][iy]) {
00160         usig[iz][iy] = GSL_NAN;
00161         vsig[iz][iy] = GSL_NAN;
00162         wsig[iz][iy] = GSL_NAN;
00163         n[iz][iy] = 0;
00164     }
00165 }
00166 }
00167
00168 /* Create output file... */
00169 LOG(1, "Write subgrid data file: %s", argv[2]);
00170 if (!out = fopen(argv[2], "w"))
00171     ERRMSG("Cannot create file!");

```

```
00172
00173 /* Write header... */
00174 fprintf(out,
00175     "# $1 = time [s]\n"
00176     "# $2 = altitude [km]\n"
00177     "# $3 = longitude [deg]\n"
00178     "# $4 = latitude [deg]\n"
00179     "# $5 = zonal wind standard deviation [m/s]\n"
00180     "# $6 = meridional wind standard deviation [m/s]\n"
00181     "# $7 = vertical velocity standard deviation [m/s]\n"
00182     "# $8 = number of data points\n");
00183
00184 /* Write output... */
00185 for (iy = 0; iy < met0->ny - 1; iy++) {
00186     fprintf(out, "\n");
00187     for (iz = 0; iz < met0->np - 1; iz++)
00188         fprintf(out, "%.2f %g %g %g %g %g %d\n",
00189             0.5 * (met0->time + met1->time),
00190             0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
00191             0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
00192             usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
00193             wsig[iz][iy] / n[iz][iy], n[iz][iy]);
00194 }
00195
00196 /* Close file... */
00197 fclose(out);
00198
00199 /* Free... */
00200 free(clim);
00201 free(met0);
00202 free(met1);
00203
00204 return EXIT_SUCCESS;
00205 }
```



Here is the call graph for this function:



## 5.36 met\_subgrid.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     clim_t *clim;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];
00044
00045     static float u[16], v[16], w[16];
00046
00047     static int i, ix, iy, iz, n[EP][EY];
00048
00049     /* Allocate... */
00050     ALLOC(clim, clim_t, 1);
00051     ALLOC(met0, met_t, 1);
00052     ALLOC(met1, met_t, 1);
00053
00054     /* Check arguments... */
00055     if (argc < 4 && argc % 2 != 0)
00056         ERRMSG
00057             ("Give parameters: <ctl> <subgrid.tab> <met0> <met1> [ <met0> <met1> ... ]");
00058
00059     /* Read control parameters... */
00060     read_ctl(argv[1], argc, argv, &ctl);
00061
00062     /* Read climatological data... */
00063     read_clim(&ctl, clim);
00064
00065     /* Loop over data files... */
00066     for (i = 3; i < argc - 1; i += 2) {
00067
00068         /* Read meteorological data... */
00069         if (!read_met(argv[i], &ctl, clim, met0))
00070             ERRMSG("Cannot open file!");
00071         if (!read_met(argv[i + 1], &ctl, clim, met1))
00072             ERRMSG("Cannot open file!");
00073
00074         /* Loop over grid boxes... */
00075         for (ix = 0; ix < met0->nx - 1; ix++)
00076             for (iy = 0; iy < met0->ny - 1; iy++)
00077                 for (iz = 0; iz < met0->np - 1; iz++) {
00078
00079                     /* Collect local wind data... */
00080                     u[0] = met0->u[ix][iy][iz];
00081                     u[1] = met0->u[ix + 1][iy][iz];
00082                     u[2] = met0->u[ix][iy + 1][iz];
00083                     u[3] = met0->u[ix + 1][iy + 1][iz];
00084                     u[4] = met0->u[ix][iy][iz + 1];
00085                     u[5] = met0->u[ix + 1][iy][iz + 1];
00086                     u[6] = met0->u[ix][iy + 1][iz + 1];
00087                     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00088
00089                     v[0] = met0->v[ix][iy][iz];
00090                     v[1] = met0->v[ix + 1][iy][iz];
00091                     v[2] = met0->v[ix][iy + 1][iz];
00092                     v[3] = met0->v[ix + 1][iy + 1][iz];
00093                     v[4] = met0->v[ix][iy][iz + 1];
00094                     v[5] = met0->v[ix + 1][iy][iz + 1];
00095                     v[6] = met0->v[ix][iy + 1][iz + 1];
00096                     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00097
00098                     w[0] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
00099                     w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
00100                     w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
00101                     w[3] =
00102                         (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
00103                     w[4] =
00104                         (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
00105                     w[5] =
00106                         (float) (1e3 *
00107                             DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
00108                     w[6] =
00109                         (float) (1e3 *
00110                             DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
00111                     w[7] =

```

```

00112         (float) (1e3 *
00113                 DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[iz + 1]));
00114
00115     /* Collect local wind data... */
00116     u[8] = met1->u[ix][iy][iz];
00117     u[9] = met1->u[ix + 1][iy][iz];
00118     u[10] = met1->u[ix][iy + 1][iz];
00119     u[11] = met1->u[ix + 1][iy + 1][iz];
00120     u[12] = met1->u[ix][iy][iz + 1];
00121     u[13] = met1->u[ix + 1][iy][iz + 1];
00122     u[14] = met1->u[ix][iy + 1][iz + 1];
00123     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00124
00125     v[8] = met1->v[ix][iy][iz];
00126     v[9] = met1->v[ix + 1][iy][iz];
00127     v[10] = met1->v[ix][iy + 1][iz];
00128     v[11] = met1->v[ix + 1][iy + 1][iz];
00129     v[12] = met1->v[ix][iy][iz + 1];
00130     v[13] = met1->v[ix + 1][iy][iz + 1];
00131     v[14] = met1->v[ix][iy + 1][iz + 1];
00132     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00133
00134     w[8] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
00135     w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
00136     w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
00137     w[11] =
00138         (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
00139     w[12] =
00140         (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
00141     w[13] =
00142         (float) (1e3 *
00143                 DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
00144     w[14] =
00145         (float) (1e3 *
00146                 DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
00147     w[15] =
00148         (float) (1e3 *
00149                 DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));
00150
00151     /* Get standard deviations of local wind data... */
00152     usig[iz][iy] += stddev(u, 16);
00153     vsig[iz][iy] += stddev(v, 16);
00154     wsig[iz][iy] += stddev(w, 16);
00155     n[iz][iy]++;
00156
00157     /* Check surface pressure... */
00158     if (met0->p[iz] > met0->ps[ix][iy]
00159         || met1->p[iz] > met1->ps[ix][iy]) {
00160         usig[iz][iy] = GSL_NAN;
00161         vsig[iz][iy] = GSL_NAN;
00162         wsig[iz][iy] = GSL_NAN;
00163         n[iz][iy] = 0;
00164     }
00165 }
00166 }
00167
00168 /* Create output file... */
00169 LOG(1, "Write subgrid data file: %s", argv[2]);
00170 if (! (out = fopen(argv[2], "w")))
00171     ERRMSG("Cannot create file!");
00172
00173 /* Write header... */
00174 fprintf(out,
00175         "# $1 = time [s]\n"
00176         "# $2 = altitude [km]\n"
00177         "# $3 = longitude [deg]\n"
00178         "# $4 = latitude [deg]\n"
00179         "# $5 = zonal wind standard deviation [m/s]\n"
00180         "# $6 = meridional wind standard deviation [m/s]\n"
00181         "# $7 = vertical velocity standard deviation [m/s]\n"
00182         "# $8 = number of data points\n");
00183
00184 /* Write output... */
00185 for (iy = 0; iy < met0->ny - 1; iy++) {
00186     fprintf(out, "\n");
00187     for (iz = 0; iz < met0->np - 1; iz++)
00188         fprintf(out, "%.2f %g %g %g %g %g %d\n",
00189                 0.5 * (met0->time + met1->time),
00190                 0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
00191                 0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
00192                 usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
00193                 wsig[iz][iy] / n[iz][iy], n[iz][iy]);
00194 }
00195
00196 /* Close file... */
00197 fclose(out);
00198

```

```
00199  /* Free... */
00200  free(clim);
00201  free(met0);
00202  free(met1);
00203
00204  return EXIT_SUCCESS;
00205 }
```

## 5.37 met\_zm.c File Reference

Extract zonal mean from meteorological data.

```
#include "libtrac.h"
```

### Macros

- `#define NZ 1000`  
*Maximum number of altitudes.*
- `#define NY 721`  
*Maximum number of latitudes.*

### Functions

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

#### 5.37.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file [met\\_zm.c](#).

#### 5.37.2 Macro Definition Documentation

##### 5.37.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line [32](#) of file [met\\_zm.c](#).

##### 5.37.2.2 NY `#define NY 721`

Maximum number of latitudes.

Definition at line [35](#) of file [met\\_zm.c](#).

## 5.37.3 Function Documentation

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

Definition at line 41 of file [met\\_zm.c](#).

```
00043     {
00044
00045     ctl_t ctl;
00046
00047     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00054         usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
00055         pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
00056         capem[NZ][NY], cinm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY],
00057         um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY],
00058         pvm[NZ][NY], o3m[NZ][NY], lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY],
00059         rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
00060         hno3m[NZ][NY], ohm[NZ][NY], h2o2m[NZ][NY], z, z0, z1, dz, zt, tt,
00061         plev[NZ], ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
00062         cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
00063         lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];
00064
00065     static int i, ix, iy, iz, np[NZ][NY], npc[NZ][NY], npt[NZ][NY], ny, nz,
00066         ci[3];
00067
00068     /* Allocate... */
00069     ALLOC(clim, clim_t, 1);
00070     ALLOC(met, met_t, 1);
00071
00072     /* Check arguments... */
00073     if (argc < 4)
00074         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00075
00076     /* Read control parameters... */
00077     read_ctl(argv[1], argc, argv, &ctl);
00078     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00079     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00080     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00081     lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
00082     lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", NULL);
00083     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00084     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00085     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00086
00087     /* Read climatological data... */
00088     read_clim(&ctl, clim);
00089
00090     /* Loop over files... */
00091     for (i = 3; i < argc; i++) {
00092
00093         /* Read meteorological data... */
00094         if (!read_met(argv[i], &ctl, clim, met))
00095             continue;
00096
00097         /* Set vertical grid... */
00098         if (z0 < 0)
00099             z0 = Z(met->p[0]);
00100         if (z1 < 0)
00101             z1 = Z(met->p[met->np - 1]);
00102         nz = 0;
00103         if (dz < 0) {
00104             for (iz = 0; iz < met->np; iz++)
00105                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00106                     plev[nz] = met->p[iz];
00107                     if (++nz > NZ)
00108                         ERRMSG("Too many pressure levels!");
00109                 }
00110             } else
00111                 for (z = z0; z <= z1; z += dz) {
00112                     plev[nz] = P(z);
00113                     if (++nz > NZ)
00114                         ERRMSG("Too many pressure levels!");
```

```

00115     }
00116
00117     /* Set horizontal grid... */
00118     if (dlat <= 0)
00119         dlat = fabs(met->lat[1] - met->lat[0]);
00120     ny = 0;
00121     if (lat0 < -90 && lat1 > 90) {
00122         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00123         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00124     }
00125     for (lat = lat0; lat <= lat1; lat += dlat) {
00126         lats[ny] = lat;
00127         if ((++ny) > NY)
00128             ERRMSG("Too many latitudes!");
00129     }
00130
00131     /* Average... */
00132     for (ix = 0; ix < met->nx; ix++)
00133         if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
00134             for (iy = 0; iy < ny; iy++)
00135                 for (iz = 0; iz < nz; iz++) {
00136
00137                     /* Interpolate meteo data... */
00138                     INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);
00139
00140                     /* Averaging... */
00141                     timem[iz][iy] += met->time;
00142                     lonm[iz][iy] += met->lon[ix];
00143                     zm[iz][iy] += z;
00144                     tm[iz][iy] += t;
00145                     um[iz][iy] += u;
00146                     vm[iz][iy] += v;
00147                     wm[iz][iy] += w;
00148                     pvm[iz][iy] += pv;
00149                     h2om[iz][iy] += h2o;
00150                     o3m[iz][iy] += o3;
00151                     lwcm[iz][iy] += lwc;
00152                     iwc[m][iz][iy] += iwc;
00153                     psm[iz][iy] += ps;
00154                     tsm[iz][iy] += ts;
00155                     zsm[iz][iy] += zs;
00156                     usm[iz][iy] += us;
00157                     vsm[iz][iy] += vs;
00158                     pblm[iz][iy] += pbl;
00159                     pctm[iz][iy] += pct;
00160                     pcbm[iz][iy] += pcb;
00161                     clm[iz][iy] += cl;
00162                     if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00163                         && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00164                         plclm[iz][iy] += plcl;
00165                         plfcm[iz][iy] += plfc;
00166                         pelm[iz][iy] += pel;
00167                         capem[iz][iy] += cape;
00168                         cinm[iz][iy] += cin;
00169                         npc[iz][iy]++;
00170                     }
00171                     if (gsl_finite(pt)) {
00172                         ptm[iz][iy] += pt;
00173                         ztm[iz][iy] += zt;
00174                         ttm[iz][iy] += tt;
00175                         h2otm[iz][iy] += h2ot;
00176                         npt[iz][iy]++;
00177                     }
00178                     rhm[iz][iy] += RH(plev[iz], t, h2o);
00179                     rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
00180                     tdewm[iz][iy] += TDEW(plev[iz], h2o);
00181                     ticem[iz][iy] += TICE(plev[iz], h2o);
00182                     hno3m[iz][iy] += clim_hno3(clim, met->time, lats[iy], plev[iz]);
00183                     tnatm[iz][iy] +=
00184                         nat_temperature(plev[iz], h2o,
00185                                         clim_hno3(clim, met->time, lats[iy], plev[iz]));
00186                     ohm[iz][iy] +=
00187                         clim_oh_diurnal(&ctl, clim, met->time, plev[iz], met->lon[ix],
00188                                         lats[iy]);
00189                     h2o2m[iz][iy] += clim_h2o2(clim, met->time, lats[iy], plev[iz]);
00190                     np[iz][iy]++;
00191                 }
00192     }
00193
00194     /* Create output file... */
00195     LOG(1, "Write meteorological data file: %s", argv[2]);
00196     if (!(out = fopen(argv[2], "w")))
00197         ERRMSG("Cannot create file!");
00198
00199     /* Write header... */
00200     fprintf(out,
00201         "# $1 = time [s]\n"

```

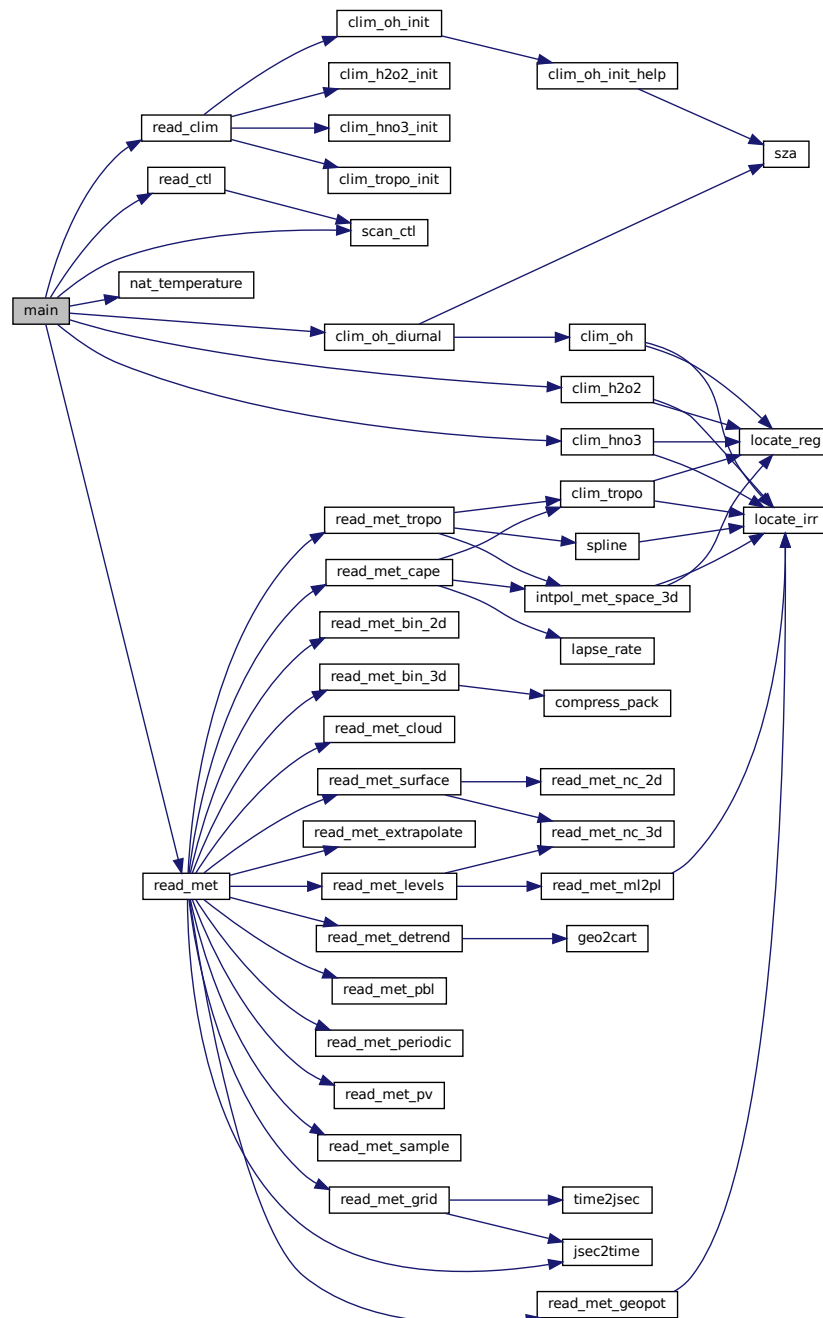
```

00202     "# $2 = altitude [km]\n"
00203     "# $3 = longitude [deg]\n"
00204     "# $4 = latitude [deg]\n"
00205     "# $5 = pressure [hPa]\n"
00206     "# $6 = temperature [K]\n"
00207     "# $7 = zonal wind [m/s]\n"
00208     "# $8 = meridional wind [m/s]\n"
00209     "# $9 = vertical velocity [hPa/s]\n"
00210     "# $10 = H2O volume mixing ratio [ppv]\n");
00211 fprintf(out,
00212     "# $11 = O3 volume mixing ratio [ppv]\n"
00213     "# $12 = geopotential height [km]\n"
00214     "# $13 = potential vorticity [PVU]\n"
00215     "# $14 = surface pressure [hPa]\n"
00216     "# $15 = surface temperature [K]\n"
00217     "# $16 = surface geopotential height [km]\n"
00218     "# $17 = surface zonal wind [m/s]\n"
00219     "# $18 = surface meridional wind [m/s]\n"
00220     "# $19 = tropopause pressure [hPa]\n"
00221     "# $20 = tropopause geopotential height [km]\n");
00222 fprintf(out,
00223     "# $21 = tropopause temperature [K]\n"
00224     "# $22 = tropopause water vapor [ppv]\n"
00225     "# $23 = cloud liquid water content [kg/kg]\n"
00226     "# $24 = cloud ice water content [kg/kg]\n"
00227     "# $25 = total column cloud water [kg/m^2]\n"
00228     "# $26 = cloud top pressure [hPa]\n"
00229     "# $27 = cloud bottom pressure [hPa]\n"
00230     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00231     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00232     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00233 fprintf(out,
00234     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00235     "# $32 = convective inhibition (CIN) [J/kg]\n"
00236     "# $33 = relative humidity over water [%]\n"
00237     "# $34 = relative humidity over ice [%]\n"
00238     "# $35 = dew point temperature [K]\n"
00239     "# $36 = frost point temperature [K]\n"
00240     "# $37 = NAT temperature [K]\n"
00241     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00242     "# $39 = OH concentration [molec/cm^3]\n"
00243     "# $40 = H2O2 concentration [molec/cm^3]\n");
00244 fprintf(out,
00245     "# $41 = boundary layer pressure [hPa]\n"
00246     "# $42 = number of data points\n"
00247     "# $43 = number of tropopause data points\n"
00248     "# $44 = number of CAPE data points\n");
00249
00250 /* Write data... */
00251 for (iz = 0; iz < nz; iz++) {
00252     fprintf(out, "\n");
00253     for (iy = 0; iy < ny; iy++)
00254         fprintf(out,
00255             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00256             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00257             " %g %g %g %g %g %d %d %d\n",
00258             timem[iz][iy] / np[iz][iy], Z(plev[iz]),
00259             lonm[iz][iy] / np[iz][iy], lats[iy],
00260             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00261             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00262             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00263             zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00264             psm[iz][iy] / np[iz][iy], tsm[iz][iy] / np[iz][iy],
00265             zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
00266             vsm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00267             ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
00268             h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00269             icwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
00270             pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],
00271             plclm[iz][iy] / npc[iz][iy], plfcm[iz][iy] / npc[iz][iy],
00272             pelm[iz][iy] / npc[iz][iy], capem[iz][iy] / npc[iz][iy],
00273             cinm[iz][iy] / npc[iz][iy], rhm[iz][iy] / np[iz][iy],
00274             rhicem[iz][iy] / np[iz][iy], tdewm[iz][iy] / np[iz][iy],
00275             ticem[iz][iy] / np[iz][iy], tnatm[iz][iy] / np[iz][iy],
00276             hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
00277             h2o2m[iz][iy] / np[iz][iy], pblm[iz][iy] / np[iz][iy],
00278             np[iz][iy], npt[iz][iy], npc[iz][iy]);
00279 }
00280
00281 /* Close file... */
00282 fclose(out);
00283
00284 /* Free... */
00285 free(clim);
00286 free(met);
00287
00288 return EXIT_SUCCESS;

```

```
00289 }
```

Here is the call graph for this function:



### 5.38 met\_zm.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.

```



```

00008
00009 MPTRAC is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU General Public License for more details.
00013
00014 You should have received a copy of the GNU General Public License
00015 along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2013-2022 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     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00054         usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
00055         pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
00056         capem[NZ][NY], cinm[NZ][NY], ttcm[NZ][NY], ztm[NZ][NY], tm[NZ][NY],
00057         um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY],
00058         pvm[NZ][NY], o3m[NZ][NY], lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY],
00059         rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
00060         hno3m[NZ][NY], ohm[NZ][NY], h2o2m[NZ][NY], z, z0, z1, dz, zt, tt,
00061         plev[NZ], ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
00062         cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
00063         lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];
00064
00065     static int i, ix, iy, iz, np[NZ][NY], npc[NZ][NY], npt[NZ][NY], ny, nz,
00066         ci[3];
00067
00068     /* Allocate... */
00069     ALLOC(clim, clim_t, 1);
00070     ALLOC(met, met_t, 1);
00071
00072     /* Check arguments... */
00073     if (argc < 4)
00074         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00075
00076     /* Read control parameters... */
00077     read_ctl(argv[1], argc, argv, &ctl);
00078     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00079     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00080     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00081     lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
00082     lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", NULL);
00083     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00084     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00085     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00086
00087     /* Read climatological data... */
00088     read_clim(&ctl, clim);
00089
00090     /* Loop over files... */
00091     for (i = 3; i < argc; i++) {
00092
00093         /* Read meteorological data... */
00094         if (!read_met(argv[i], &ctl, clim, met))
00095             continue;
00096
00097         /* Set vertical grid... */
00098         if (z0 < 0)
00099             z0 = Z(met->p[0]);
00100         if (z1 < 0)
00101             z1 = Z(met->p[met->np - 1]);

```

```

00102     nz = 0;
00103     if (dz < 0) {
00104         for (iz = 0; iz < met->np; iz++)
00105             if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00106                 plev[nz] = met->p[iz];
00107                 if ((++nz) > NZ)
00108                     ERRMSG("Too many pressure levels!");
00109             }
00110     } else
00111         for (z = z0; z <= z1; z += dz) {
00112             plev[nz] = P(z);
00113             if ((++nz) > NZ)
00114                 ERRMSG("Too many pressure levels!");
00115         }
00116
00117     /* Set horizontal grid... */
00118     if (dlat <= 0)
00119         dlat = fabs(met->lat[1] - met->lat[0]);
00120     ny = 0;
00121     if (lat0 < -90 && lat1 > 90) {
00122         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00123         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00124     }
00125     for (lat = lat0; lat <= lat1; lat += dlat) {
00126         lats[ny] = lat;
00127         if ((++ny) > NY)
00128             ERRMSG("Too many latitudes!");
00129     }
00130
00131     /* Average... */
00132     for (ix = 0; ix < met->nx; ix++)
00133         if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
00134             for (iy = 0; iy < ny; iy++)
00135                 for (iz = 0; iz < nz; iz++) {
00136
00137                     /* Interpolate meteo data... */
00138                     INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);
00139
00140                     /* Averaging... */
00141                     timem[iz][iy] += met->time;
00142                     lonm[iz][iy] += met->lon[ix];
00143                     zm[iz][iy] += z;
00144                     tm[iz][iy] += t;
00145                     um[iz][iy] += u;
00146                     vm[iz][iy] += v;
00147                     wm[iz][iy] += w;
00148                     pvm[iz][iy] += pv;
00149                     h2om[iz][iy] += h2o;
00150                     o3m[iz][iy] += o3;
00151                     lwcm[iz][iy] += lwc;
00152                     iwcm[iz][iy] += iwc;
00153                     psm[iz][iy] += ps;
00154                     tsm[iz][iy] += ts;
00155                     zsm[iz][iy] += zs;
00156                     usm[iz][iy] += us;
00157                     vsm[iz][iy] += vs;
00158                     pblm[iz][iy] += pbl;
00159                     pctm[iz][iy] += pct;
00160                     pcbm[iz][iy] += pcb;
00161                     clm[iz][iy] += cl;
00162                     if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00163                         && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00164                         plclm[iz][iy] += plcl;
00165                         plfcm[iz][iy] += plfc;
00166                         pelm[iz][iy] += pel;
00167                         capem[iz][iy] += cape;
00168                         cinm[iz][iy] += cin;
00169                         npc[iz][iy]++;
00170                     }
00171                     if (gsl_finite(pt)) {
00172                         ptm[iz][iy] += pt;
00173                         ztm[iz][iy] += zt;
00174                         ttm[iz][iy] += tt;
00175                         h2otm[iz][iy] += h2ot;
00176                         npt[iz][iy]++;
00177                     }
00178                     rhm[iz][iy] += RH(plev[iz], t, h2o);
00179                     rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
00180                     tdewm[iz][iy] += TDEW(plev[iz], h2o);
00181                     ticem[iz][iy] += TICE(plev[iz], h2o);
00182                     hno3m[iz][iy] += clim_hno3(clim, met->time, lats[iy], plev[iz]);
00183                     tnatm[iz][iy] +=
00184                         nat_temperature(plev[iz], h2o,
00185                                         clim_hno3(clim, met->time, lats[iy], plev[iz]));
00186                     ohm[iz][iy] +=
00187                         clim_oh_diurnal(&ctl, clim, met->time, plev[iz], met->lon[ix],
00188                                         lats[iy]);

```

```
00189 h2o2m[iz][iy] += clim_h2o2(clim, met->time, lats[iy], plev[iz]);  
00190 np[iz][iy]++;  
00191 }  
00192 }  
00193  
00194 /* Create output file... */  
00195 LOG(1, "Write meteorological data file: %s", argv[2]);  
00196 if (!out = fopen(argv[2], "w"))  
00197     ERRMSG("Cannot create file!");  
00198  
00199 /* Write header... */  
00200 fprintf(out,  
00201     "# $1 = time [s]\n"  
00202     "# $2 = altitude [km]\n"  
00203     "# $3 = longitude [deg]\n"  
00204     "# $4 = latitude [deg]\n"  
00205     "# $5 = pressure [hPa]\n"  
00206     "# $6 = temperature [K]\n"  
00207     "# $7 = zonal wind [m/s]\n"  
00208     "# $8 = meridional wind [m/s]\n"  
00209     "# $9 = vertical velocity [hPa/s]\n"  
00210     "# $10 = H2O volume mixing ratio [ppv]\n");  
00211 fprintf(out,  
00212     "# $11 = O3 volume mixing ratio [ppv]\n"  
00213     "# $12 = geopotential height [km]\n"  
00214     "# $13 = potential vorticity [PVU]\n"  
00215     "# $14 = surface pressure [hPa]\n"  
00216     "# $15 = surface temperature [K]\n"  
00217     "# $16 = surface geopotential height [km]\n"  
00218     "# $17 = surface zonal wind [m/s]\n"  
00219     "# $18 = surface meridional wind [m/s]\n"  
00220     "# $19 = tropopause pressure [hPa]\n"  
00221     "# $20 = tropopause geopotential height [km]\n");  
00222 fprintf(out,  
00223     "# $21 = tropopause temperature [K]\n"  
00224     "# $22 = tropopause water vapor [ppv]\n"  
00225     "# $23 = cloud liquid water content [kg/kg]\n"  
00226     "# $24 = cloud ice water content [kg/kg]\n"  
00227     "# $25 = total column cloud water [kg/m^2]\n"  
00228     "# $26 = cloud top pressure [hPa]\n"  
00229     "# $27 = cloud bottom pressure [hPa]\n"  
00230     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"  
00231     "# $29 = pressure at level of free convection (LFC) [hPa]\n"  
00232     "# $30 = pressure at equilibrium level (EL) [hPa]\n");  
00233 fprintf(out,  
00234     "# $31 = convective available potential energy (CAPE) [J/kg]\n"  
00235     "# $32 = convective inhibition (CIN) [J/kg]\n"  
00236     "# $33 = relative humidity over water [%]\n"  
00237     "# $34 = relative humidity over ice [%]\n"  
00238     "# $35 = dew point temperature [K]\n"  
00239     "# $36 = frost point temperature [K]\n"  
00240     "# $37 = NAT temperature [K]\n"  
00241     "# $38 = HNO3 volume mixing ratio [ppv]\n"  
00242     "# $39 = OH concentration [molec/cm^3]\n"  
00243     "# $40 = H2O2 concentration [molec/cm^3]\n");  
00244 fprintf(out,  
00245     "# $41 = boundary layer pressure [hPa]\n"  
00246     "# $42 = number of data points\n"  
00247     "# $43 = number of tropopause data points\n"  
00248     "# $44 = number of CAPE data points\n");  
00249  
00250 /* Write data... */  
00251 for (iz = 0; iz < nz; iz++) {  
00252     fprintf(out, "\n");  
00253     for (iy = 0; iy < ny; iy++)  
00254         fprintf(out,  
00255             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g "  
00256             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g "  
00257             " %g %g %g %g %d %d %d\n",  
00258             timem[iz][iy] / np[iz][iy], Z(plev[iz]),  
00259             lonm[iz][iy] / np[iz][iy], lats[iy],  
00260             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],  
00261             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],  
00262             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],  
00263             zm[iz][iy] / np[iz][iy], pvmm[iz][iy] / np[iz][iy],  
00264             psbm[iz][iy] / np[iz][iy], tsm[iz][iy] / np[iz][iy],  
00265             zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],  
00266             vsbm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],  
00267             ztm[iz][iy] / npt[iz][iy], ttcm[iz][iy] / npt[iz][iy],  
00268             h2otc[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],  
00269             iwcmm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],  
00270             pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],  
00271             plclm[iz][iy] / npc[iz][iy], plfcmm[iz][iy] / npc[iz][iy],  
00272             pelm[iz][iy] / npc[iz][iy], capemm[iz][iy] / npc[iz][iy],  
00273             cinm[iz][iy] / npc[iz][iy], rhm[iz][iy] / np[iz][iy],  
00274             rhicem[iz][iy] / np[iz][iy], tdewmm[iz][iy] / np[iz][iy],  
00275             ticem[iz][iy] / np[iz][iy], tnatmm[iz][iy] / np[iz][iy]
```

```

00276             hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
00277             h2o2m[iz][iy] / np[iz][iy], pblm[iz][iy] / np[iz][iy],
00278             np[iz][iy], npt[iz][iy], npc[iz][iy]);
00279     }
00280
00281     /* Close file... */
00282     fclose(out);
00283
00284     /* Free... */
00285     free(clim);
00286     free(met);
00287
00288     return EXIT_SUCCESS;
00289 }

```

## 5.39 sedi.c File Reference

Calculate sedimentation velocity.

```
#include "libtrac.h"
```

### Functions

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

#### 5.39.1 Detailed Description

Calculate sedimentation velocity.

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

#### 5.39.2 Function Documentation

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

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

```

00029     {
00030
00031     double eta, p, T, r_p, rho, Re, rho_p, vs;
00032
00033     /* Check arguments... */
00034     if (argc < 5)
00035         ERRMSG("Give parameters: <p> <T> <r_p> <rho_p>");
00036
00037     /* Read arguments... */
00038     p = atof(argv[1]);
00039     T = atof(argv[2]);
00040     r_p = atof(argv[3]);
00041     rho_p = atof(argv[4]);
00042
00043     /* Calculate sedimentation velocity... */
00044     vs = sedi(p, T, r_p, rho_p);
00045
00046     /* Density of dry air [kg / m^3]... */
00047     rho = 100. * p / (RA * T);
00048
00049     /* Dynamic viscosity of air [kg / (m s)]... */

```

```

00050     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00051
00052     /* Particle Reynolds number... */
00053     Re = 2e-6 * r_p * vs * rho / eta;
00054
00055     /* Write output... */
00056     printf("    p= %g hPa\n", p);
00057     printf("    T= %g K\n", T);
00058     printf("    r_p= %g microns\n", r_p);
00059     printf("rho_p= %g kg/m^3\n", rho_p);
00060     printf("rho_a= %g kg/m^3\n", RHO(p, T));
00061     printf("    v_s= %g m/s\n", vs);
00062     printf("    Re= %g\n", Re);
00063
00064     return EXIT_SUCCESS;
00065 }

```

Here is the call graph for this function:



## 5.40 sedi.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     double eta, p, T, r_p, rho, Re, rho_p, vs;
00032
00033     /* Check arguments... */
00034     if (argc < 5)
00035         ERRMSG("Give parameters: <p> <T> <r_p> <rho_p>");
00036
00037     /* Read arguments... */
00038     p = atof(argv[1]);
00039     T = atof(argv[2]);
00040     r_p = atof(argv[3]);
00041     rho_p = atof(argv[4]);
00042
00043     /* Calculate sedimentation velocity... */
00044     vs = sedi(p, T, r_p, rho_p);
00045
00046     /* Density of dry air [kg / m^3]... */
00047     rho = 100. * p / (RA * T);
00048
00049     /* Dynamic viscosity of air [kg / (m s)]... */
00050     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00051
00052     /* Particle Reynolds number... */

```

```
00053 Re = 2e-6 * r_p * vs * rho / eta;
00054
00055 /* Write output... */
00056 printf("    p= %g hPa\n", p);
00057 printf("    T= %g K\n", T);
00058 printf("    r_p= %g microns\n", r_p);
00059 printf("rho_p= %g kg/m^3\n", rho_p);
00060 printf("rho_a= %g kg/m^3\n", RHO(p, T));
00061 printf("    v_s= %g m/s\n", vs);
00062 printf("    Re= %g\n", Re);
00063
00064 return EXIT_SUCCESS;
00065 }
```

## 5.41 time2jsec.c File Reference

Convert date to Julian seconds.

```
#include "libtrac.h"
```

### Functions

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

#### 5.41.1 Detailed Description

Convert date to Julian seconds.

Definition in file [time2jsec.c](#).

#### 5.41.2 Function Documentation

**5.41.2.1 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:



## 5.42 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
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     double jsec, remain;
00032
00033     int day, hour, min, mon, sec, year;
00034
00035     /* Check arguments... */
00036     if (argc < 8)
00037         ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039     /* Read arguments... */
00040     year = atoi(argv[1]);
00041     mon = atoi(argv[2]);
00042     day = atoi(argv[3]);
00043     hour = atoi(argv[4]);
00044     min = atoi(argv[5]);
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 }

```

## 5.43 tnat.c File Reference

Calculate PSC temperatures.

```
#include "libtrac.h"
```

## Functions

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

### 5.43.1 Detailed Description

Calculate PSC temperatures.

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

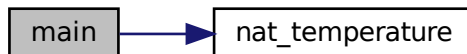
### 5.43.2 Function Documentation

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

Definition at line 31 of file [tnat.c](#).

```
00033     {  
00034  
00035     /* Check arguments... */  
00036     if (argc < 3)  
00037         ERRMSG("Give parameters: <p> <h2o> <hno3>");  
00038  
00039     /* Get variables... */  
00040     double p = atof(argv[1]);  
00041     double h2o = atof(argv[2]);  
00042     double hno3 = atof(argv[3]);  
00043  
00044     /* Calculate T_ice and T_NAT... */  
00045     double tice = TICE(p, h2o);  
00046     double tnat = nat_temperature(p, h2o, hno3);  
00047  
00048     /* Write output... */  
00049     printf("      p= %g hPa\n", p);  
00050     printf(" q_H2O= %g ppv\n", h2o);  
00051     printf(" q_HNO3= %g ppv\n", hno3);  
00052     printf(" T_ice= %g K\n", tice);  
00053     printf(" T_NAT= %g K\n", tnat);  
00054  
00055     return EXIT_SUCCESS;  
00056 }
```

Here is the call graph for this function:





## 5.44 tnat.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     /* Check arguments... */
00036     if (argc < 3)
00037         ERRMSG("Give parameters: <p> <h2o> <hno3>");
00038
00039     /* Get variables... */
00040     double p = atof(argv[1]);
00041     double h2o = atof(argv[2]);
00042     double hno3 = atof(argv[3]);
00043
00044     /* Calculate T_ice and T_NAT... */
00045     double tice = TICE(p, h2o);
00046     double tnat = nat_temperature(p, h2o, hno3);
00047
00048     /* Write output... */
00049     printf("      p= %g hPa\n", p);
00050     printf(" q_H2O= %g ppv\n", h2o);
00051     printf(" q_HNO3= %g ppv\n", hno3);
00052     printf(" T_ice= %g K\n", tice);
00053     printf(" T_NAT= %g K\n", tnat);
00054
00055     return EXIT_SUCCESS;
00056 }

```

## 5.45 trac.c File Reference

Lagrangian particle dispersion model.

```
#include "libtrac.h"
```

### Functions

- void `module_advect` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate advection of air parcels.*
- void `module_bound_cond` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Apply boundary conditions.*
- 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, `clim_t` \*clim, `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, `clim_t` \*clim, `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, `clim_t` \*clim, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm)  
*Interpolate meteo data for air parcel positions.*
- void `module_oh_chem` (`ctl_t` \*ctl, `clim_t` \*clim, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Calculate OH chemistry.*
- void `module_h2o2_chem` (`ctl_t` \*ctl, `clim_t` \*clim, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt, double \*rs)  
*Calculate H2O2 chemistry.*
- void `module_position` (`ctl_t` \*ctl, `met_t` \*met0, `met_t` \*met1, `atm_t` \*atm, double \*dt)  
*Check position of air parcels.*
- void `module_rng_init` (int ntask)  
*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_sort` (`ctl_t` \*ctl, `met_t` \*met0, `atm_t` \*atm)  
*Sort particles according to box index.*
- void `module_sort_help` (double \*a, int \*p, int np)  
*Helper function for sorting module.*
- void `module_timesteps` (`ctl_t` \*ctl, `atm_t` \*atm, double \*dt, double t)  
*Calculate time steps.*
- void `module_timesteps_init` (`ctl_t` \*ctl, `atm_t` \*atm)  
*Initialize timesteps.*
- 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[])

### 5.45.1 Detailed Description

Lagrangian particle dispersion model.

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

### 5.45.2 Function Documentation

**5.45.2.1 module\_advect()** void module\_advect (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate advection of air parcels.

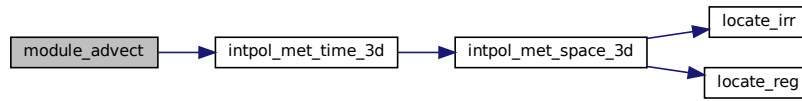
Definition at line 545 of file trac.c.

```

00550     {
00551
00552     /* Set timer... */
00553     SELECT_TIMER("MODULE_ADEVECTION", "PHYSICS", NVTX_GPU);
00554
00555     const int np = atm->np;
00556 #ifdef _OPENACC
00557 #pragma acc data present(ctl,met0,met1,atm,dt)
00558 #pragma acc parallel loop independent gang vector
00559 #else
00560 #pragma omp parallel for default(shared)
00561 #endif
00562     for (int ip = 0; ip < np; ip++)
00563         if (dt[ip] != 0) {
00564
00565             /* Init... */
00566             double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];
00567
00568             /* Loop over integration nodes... */
00569             for (int i = 0; i < ctl->advect; i++) {
00570
00571                 /* Set position... */
00572                 if (i == 0) {
00573                     dts = 0.0;
00574                     x[0] = atm->lon[ip];
00575                     x[1] = atm->lat[ip];
00576                     x[2] = atm->p[ip];
00577                 } else {
00578                     dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
00579                     x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
00580                     x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
00581                     x[2] = atm->p[ip] + dts * w[i - 1];
00582                 }
00583                 double tm = atm->time[ip] + dts;
00584
00585                 /* Interpolate meteo data... */
00586 #ifdef UVW
00587                 intpol_met_time_uvw(met0, met1, tm, x[2], x[0], x[1],
00588                                     &u[i], &v[i], &w[i]);
00589 #else
00590                 INTPOL_INIT;
00591                 intpol_met_time_3d(met0, met0->u, met1, met1->u, tm,
00592                                   x[2], x[0], x[1], &u[i], ci, cw, 1);
00593                 intpol_met_time_3d(met0, met0->v, met1, met1->v, tm,
00594                                   x[2], x[0], x[1], &v[i], ci, cw, 0);
00595                 intpol_met_time_3d(met0, met0->w, met1, met1->w, tm,
00596                                   x[2], x[0], x[1], &w[i], ci, cw, 0);
00597 #endif
00598
00599                 /* Get mean wind... */
00600                 double k = 1.0;
00601                 if (ctl->advect == 2)
00602                     k = (i == 0 ? 0.0 : 1.0);
00603                 else if (ctl->advect == 4)
00604                     k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
00605                 um += k * u[i];
00606                 vm += k * v[i];
00607                 wm += k * w[i];
00608             }
00609
00610             /* Set new position... */
00611             atm->time[ip] += dt[ip];
00612             atm->lon[ip] += DX2DEG(dt[ip] * um / 1000.,
00613                                   (ctl->advect == 2 ? x[1] : atm->lat[ip]));
00614             atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
00615             atm->p[ip] += dt[ip] * wm;
00616         }
00617 }

```

Here is the call graph for this function:



**5.45.2.2 module\_bound\_cond()** void module\_bound\_cond (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Apply boundary conditions.

Definition at line 621 of file [trac.c](#).

```

00626     {
00627
00628     /* Set timer... */
00629     SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);
00630
00631     /* Check quantity flags... */
00632     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00633         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00634
00635     const int np = atm->np;
00636 #ifdef _OPENACC
00637 #pragma acc data present(ctl, met0, met1, atm, dt)
00638 #pragma acc parallel loop independent gang vector
00639 #else
00640 #pragma omp parallel for default(shared)
00641 #endif
00642     for (int ip = 0; ip < np; ip++)
00643         if (dt[ip] != 0) {
00644
00645             double ps;
00646
00647             /* Check latitude and pressure range... */
00648             if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
00649                 || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
00650                 continue;
00651
00652             /* Check surface layer... */
00653             if (ctl->bound_dps > 0) {
00654
00655                 /* Get surface pressure... */
00656                 INTPOL_INIT;
00657                 INTPOL_2D(ps, 1);
00658
00659                 /* Check whether particle is above the surface layer... */
00660                 if (atm->p[ip] < ps - ctl->bound_dps)
00661                     continue;
00662             }
00663
00664             /* Set mass and volume mixing ratio... */
00665             if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
00666                 atm->q[ctl->qnt_m][ip] =
00667                     ctl->bound_mass + ctl->bound_mass_trend * atm->time[ip];
00668             if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
00669                 atm->q[ctl->qnt_vmr][ip] =
00670                     ctl->bound_vmr + ctl->bound_vmr_trend * atm->time[ip];
00671         }
00672     }

```

**5.45.2.3 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 676 of file [trac.c](#).

```

00682     {
00683
00684     /* Set timer... */
00685     SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);
00686
00687     /* Create random numbers... */
00688     module_rng(rs, (size_t) atm->np, 0);
00689
00690     const int np = atm->np;
00691 #ifndef _OPENACC
00692 #pragma acc data present(ctl, met0, met1, atm, dt, rs)
00693 #pragma acc parallel loop independent gang vector
00694 #else
00695 #pragma omp parallel for default(shared)
00696 #endif
00697     for (int ip = 0; ip < np; ip++)
00698         if (dt[ip] != 0) {
00699             double cape, cin, pel, ps;
00700
00701             /* Interpolate CAPE... */
00702             INTPOL_INIT;
00703             INTPOL_2D(cape, 1);
00704
00705             /* Check threshold... */
00706             if (isfinite(cape) && cape >= ctl->conv_cape) {
00707
00708                 /* Check CIN... */
00709                 if (ctl->conv_cin > 0) {
00710                     INTPOL_2D(cin, 0);
00711                     if (isfinite(cin) && cin >= ctl->conv_cin)
00712                         continue;
00713                 }
00714
00715                 /* Interpolate equilibrium level... */
00716                 INTPOL_2D(pel, 0);
00717
00718                 /* Check whether particle is above cloud top... */
00719                 if (!isfinite(pel) || atm->p[ip] < pel)
00720                     continue;
00721
00722                 /* Set pressure range for mixing... */
00723                 double pbot = atm->p[ip];
00724                 double ptop = atm->p[ip];
00725                 if (ctl->conv_mix_bot == 1) {
00726                     INTPOL_2D(ps, 0);
00727                     pbot = ps;
00728                 }
00729                 if (ctl->conv_mix_top == 1)
00730                     ptop = pel;
00731
00732                 /* Limit vertical velocity... */
00733                 if (ctl->conv_wmax > 0 || ctl->conv_wcape) {
00734                     double z = Z(atm->p[ip]);
00735                     double wmax = (ctl->conv_wcape) ? sqrt(2. * cape) : ctl->conv_wmax;
00736                     double pmax = P(z - wmax * dt[ip] / 1000.);
00737                     double pmin = P(z + wmax * dt[ip] / 1000.);
00738                     ptop = GSL_MAX(ptop, pmin);
00739                     pbot = GSL_MIN(pbot, pmax);
00740                 }
00741
00742                 /* Vertical mixing based on pressure... */
00743                 if (ctl->conv_mix == 0)
00744                     atm->p[ip] = pbot + (ptop - pbot) * rs[ip];
00745
00746                 /* Vertical mixing based on density... */
00747                 else if (ctl->conv_mix == 1) {
00748                     /* Get density range... */
00749                     double tbot, ttop;
00750                     intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00751

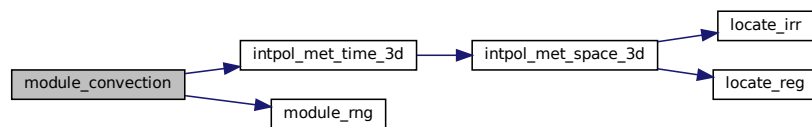
```

```

00753         pbot, atm->lon[ip], atm->lat[ip], &tbot,
00754         ci, cw, 1);
00755     intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00756         ptop, atm->lon[ip], atm->lat[ip], &ttop,
00757         ci, cw, 1);
00758     double rhobot = pbot / tbot;
00759     double rhotop = ptop / ttop;
00760
00761     /* Get new density... */
00762     double lrho = log(rhobot + (rhotop - rhobot) * rs[ip]);
00763
00764     /* Find pressure... */
00765     double lrhobot = log(rhobot);
00766     double lrhotop = log(rhotop);
00767     double lpbot = log(pbot);
00768     double lptop = log(ptop);
00769     atm->p[ip] = exp(LIN(lrhobot, lpbot, lrhotop, lptop, lrho));
00770 }
00771 }
00772 }
00773 }

```

Here is the call graph for this function:



**5.45.2.4 module\_decay()** void module\_decay (

```

    ctl_t * ctl,
    clim_t * clim,
    atm_t * atm,
    double * dt )

```

Calculate exponential decay of particle mass.

Definition at line 777 of file `trac.c`.

```

00781     {
00782
00783     /* Set timer... */
00784     SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);
00785
00786     /* Check quantity flags... */
00787     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00788         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00789
00790     const int np = atm->np;
00791     #ifdef _OPENACC
00792     #pragma acc data present(ctl,clim,atm,dt)
00793     #pragma acc parallel loop independent gang vector
00794     #else
00795     #pragma omp parallel for default(shared)
00796     #endif
00797     for (int ip = 0; ip < np; ip++)
00798         if (dt[ip] != 0) {
00799
00800         /* Get weighting factor... */
00801         double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00802
00803         /* Set lifetime... */
00804         double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00805
00806         /* Calculate exponential decay... */
00807         double aux = exp(-dt[ip] / tdec);
00808         if (ctl->qnt_m >= 0) {

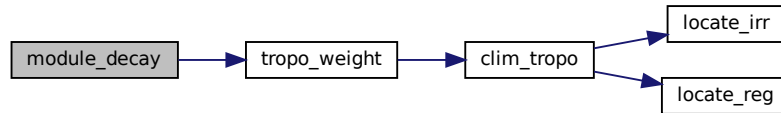
```

```

00809         if (ctl->qnt_mloss_decay >= 0)
00810             atm->q[ctl->qnt_mloss_decay][ip]
00811             += atm->q[ctl->qnt_m][ip] * (1 - aux);
00812         atm->q[ctl->qnt_m][ip] *= aux;
00813     }
00814     if (ctl->qnt_vmr >= 0)
00815         atm->q[ctl->qnt_vmr][ip] *= aux;
00816     }
00817 }

```

Here is the call graph for this function:



#### 5.45.2.5 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 821 of file trac.c.

```

00828     {
00829
00830         /* Set timer... */
00831         SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);
00832
00833         /* Create random numbers... */
00834         module_rng(rs, 3 * (size_t) atm->np, 1);
00835
00836         const int np = atm->np;
00837 #ifdef _OPENACC
00838 #pragma acc data present(ctl, met0, met1, atm, cache, dt, rs)
00839 #pragma acc parallel loop independent gang vector
00840 #else
00841 #pragma omp parallel for default(shared)
00842 #endif
00843         for (int ip = 0; ip < np; ip++)
00844             if (dt[ip] != 0) {
00845
00846                 /* Get indices... */
00847                 int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00848                 int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00849                 int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00850
00851                 /* Get standard deviations of local wind data... */
00852                 float umean = 0, usig = 0, vmean = 0, vsig = 0, wmean = 0, wsig = 0;
00853                 for (int i = 0; i < 2; i++)
00854                     for (int j = 0; j < 2; j++)
00855                         for (int k = 0; k < 2; k++) {
00856 #ifdef UVW
00857                             umean += met0->uvw[ix + i][iy + j][iz + k][0];
00858                             usig += SQR(met0->uvw[ix + i][iy + j][iz + k][0]);
00859                             vmean += met0->uvw[ix + i][iy + j][iz + k][1];
00860                             vsig += SQR(met0->uvw[ix + i][iy + j][iz + k][1]);
00861                             wmean += met0->uvw[ix + i][iy + j][iz + k][2];

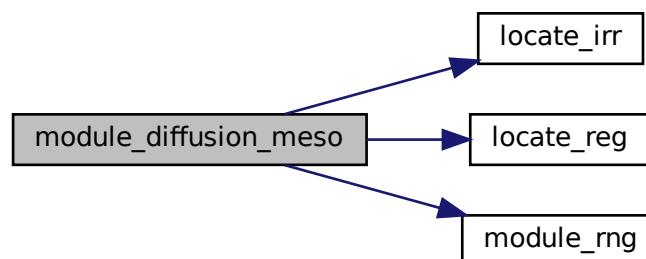
```

```

00862         wsig += SQR(met0->uvw[ix + i][iy + j][iz + k][2]);
00863
00864         umean += met1->uvw[ix + i][iy + j][iz + k][0];
00865         usig += SQR(met1->uvw[ix + i][iy + j][iz + k][0]);
00866         vmean += met1->uvw[ix + i][iy + j][iz + k][1];
00867         vsig += SQR(met1->uvw[ix + i][iy + j][iz + k][1]);
00868         wmean += met1->uvw[ix + i][iy + j][iz + k][2];
00869         wsig += SQR(met1->uvw[ix + i][iy + j][iz + k][2]);
00870     #else
00871         umean += met0->u[ix + i][iy + j][iz + k];
00872         usig += SQR(met0->u[ix + i][iy + j][iz + k]);
00873         vmean += met0->v[ix + i][iy + j][iz + k];
00874         vsig += SQR(met0->v[ix + i][iy + j][iz + k]);
00875         wmean += met0->w[ix + i][iy + j][iz + k];
00876         wsig += SQR(met0->w[ix + i][iy + j][iz + k]);
00877
00878         umean += met1->u[ix + i][iy + j][iz + k];
00879         usig += SQR(met1->u[ix + i][iy + j][iz + k]);
00880         vmean += met1->v[ix + i][iy + j][iz + k];
00881         vsig += SQR(met1->v[ix + i][iy + j][iz + k]);
00882         wmean += met1->w[ix + i][iy + j][iz + k];
00883         wsig += SQR(met1->w[ix + i][iy + j][iz + k]);
00884     #endif
00885 }
00886 usig = usig / 16.f - SQR(umean / 16.f);
00887 usig = (usig > 0 ? sqrtf(usig) : 0);
00888 vsig = vsig / 16.f - SQR(vmean / 16.f);
00889 vsig = (vsig > 0 ? sqrtf(vsig) : 0);
00890 wsig = wsig / 16.f - SQR(wmean / 16.f);
00891 wsig = (wsig > 0 ? sqrtf(wsig) : 0);
00892
00893 /* Set temporal correlations for mesoscale fluctuations... */
00894 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00895 double r2 = sqrt(1 - r * r);
00896
00897 /* Calculate horizontal mesoscale wind fluctuations... */
00898 if (ctl->turb_mesox > 0) {
00899     cache->uvw[ip][0] =
00900         (float) (r * cache->uvw[ip][0] +
00901             r2 * rs[3 * ip] * ctl->turb_mesox * usig);
00902     atm->lon[ip] +=
00903         DX2DEG(cache->uvw[ip][0] * dt[ip] / 1000., atm->lat[ip]);
00904
00905     cache->uvw[ip][1] =
00906         (float) (r * cache->uvw[ip][1] +
00907             r2 * rs[3 * ip + 1] * ctl->turb_mesox * vsig);
00908     atm->lat[ip] +=
00909         DY2DEG(cache->uvw[ip][1] * dt[ip] / 1000.);
00910 }
00911
00912 /* Calculate vertical mesoscale wind fluctuations... */
00913 if (ctl->turb_mesoz > 0) {
00914     cache->uvw[ip][2] =
00915         (float) (r * cache->uvw[ip][2] +
00916             r2 * rs[3 * ip + 2] * ctl->turb_mesoz * wsig);
00917     atm->p[ip] += cache->uvw[ip][2] * dt[ip];
00918 }
00919 }

```

Here is the call graph for this function:





### 5.45.2.6 module\_diffusion\_turb() void module\_diffusion\_turb (

```

    ctl_t * ctl,
    clim_t * clim,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate turbulent diffusion.

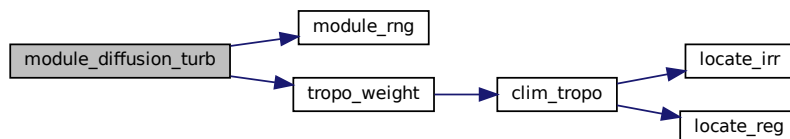
Definition at line 923 of file [trac.c](#).

```

00928     {
00929
00930     /* Set timer... */
00931     SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);
00932
00933     /* Create random numbers... */
00934     module_rng(rs, 3 * (size_t) atm->np, 1);
00935
00936     const int np = atm->np;
00937 #ifdef _OPENACC
00938 #pragma acc data present(ctl,clim,atm,dt,rs)
00939 #pragma acc parallel loop independent gang vector
00940 #else
00941 #pragma omp parallel for default(shared)
00942 #endif
00943     for (int ip = 0; ip < np; ip++)
00944         if (dt[ip] != 0) {
00945
00946             /* Get weighting factor... */
00947             double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00948
00949             /* Set diffusivity... */
00950             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00951             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00952
00953             /* Horizontal turbulent diffusion... */
00954             if (dx > 0) {
00955                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00956                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00957                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00958             }
00959
00960             /* Vertical turbulent diffusion... */
00961             if (dz > 0) {
00962                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00963                 atm->p[ip] += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00964             }
00965         }
00966     }

```

Here is the call graph for this function:



```

5.45.2.7 module_dry_deposition() void module_dry_deposition (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate dry deposition.

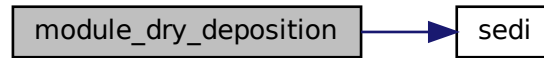
Definition at line 970 of file [trac.c](#).

```

00975     {
00976
00977     /* Set timer... */
00978     SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);
00979
00980     /* Depth of the surface layer [hPa]. */
00981     const double dp = 30.;
00982
00983     /* Check quantity flags... */
00984     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00985         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00986
00987     const int np = atm->np;
00988     #ifdef _OPENACC
00989     #pragma acc data present(ctl, met0, met1, atm, dt)
00990     #pragma acc parallel loop independent gang vector
00991     #else
00992     #pragma omp parallel for default(shared)
00993     #endif
00994     for (int ip = 0; ip < np; ip++)
00995         if (dt[ip] != 0) {
00996
00997             double ps, t, v_dep;
00998
00999             /* Get surface pressure... */
01000             INTPOL_INIT;
01001             INTPOL_2D(ps, 1);
01002
01003             /* Check whether particle is above the surface layer... */
01004             if (atm->p[ip] < ps - dp)
01005                 continue;
01006
01007             /* Set depth of surface layer... */
01008             double dz = 1000. * (Z(ps - dp) - Z(ps));
01009
01010             /* Calculate sedimentation velocity for particles... */
01011             if (ctl->qnt_rp > 0 && ctl->qnt_rhop > 0) {
01012
01013                 /* Get temperature... */
01014                 INTPOL_3D(t, 1);
01015
01016                 /* Set deposition velocity... */
01017                 v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01018                             atm->q[ctl->qnt_rhop][ip]);
01019             }
01020
01021             /* Use explicit sedimentation velocity for gases... */
01022             else
01023                 v_dep = ctl->dry_depo[0];
01024
01025             /* Calculate loss of mass based on deposition velocity... */
01026             double aux = exp(-dt[ip] * v_dep / dz);
01027             if (ctl->qnt_m >= 0) {
01028                 if (ctl->qnt_mloss_dry >= 0)
01029                     atm->q[ctl->qnt_mloss_dry][ip]
01030                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01031                 atm->q[ctl->qnt_m][ip] *= aux;
01032             }
01033             if (ctl->qnt_vmr >= 0)
01034                 atm->q[ctl->qnt_vmr][ip] *= aux;
01035         }
01036 }

```

Here is the call graph for this function:



#### 5.45.2.8 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 1040 of file trac.c.

```

01045     {
01046
01047     FILE *in;
01048
01049     char line[LEN];
01050
01051     double t;
01052
01053     /* Set timer... */
01054     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01055
01056     /* Init... */
01057     INTPOL_INIT;
01058
01059     /* Save pressure... */
01060     if (ctl->isosurf == 1)
01061         for (int ip = 0; ip < atm->np; ip++) {
01062             cache->iso_var[ip] = atm->p[ip];
01063         }
01064     /* Save density... */
01065     else if (ctl->isosurf == 2)
01066         for (int ip = 0; ip < atm->np; ip++) {
01067             INTPOL_3D(t, 1);
01068             cache->iso_var[ip] = atm->p[ip] / t;
01069         }
01070
01071     /* Save potential temperature... */
01072     else if (ctl->isosurf == 3)
01073         for (int ip = 0; ip < atm->np; ip++) {
01074             INTPOL_3D(t, 1);
01075             cache->iso_var[ip] = THETA(atm->p[ip], t);
01076         }
01077
01078     /* Read balloon pressure data... */
01079     else if (ctl->isosurf == 4) {
01080
01081         /* Write info... */
01082         LOG(1, "Read balloon pressure data: %s", ctl->balloon);
01083
01084         /* Open file... */
01085         if (!(in = fopen(ctl->balloon, "r")))
01086             ERRMSG("Cannot open file!");
01087
01088         /* Read pressure time series... */
01089         while (fgets(line, LEN, in))
01090             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
01091                       &(cache->iso_ps[cache->iso_n])) == 2)
01092                 if (++cache->iso_n > NP)

```

```

01093         ERRMSG("Too many data points!");
01094
01095     /* Check number of points... */
01096     if (cache->iso_n < 1)
01097         ERRMSG("Could not read any data!");
01098
01099     /* Close file... */
01100     fclose(in);
01101 }
01102 }

```

**5.45.2.9 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.

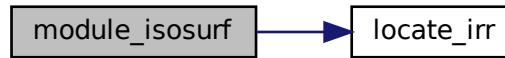
Definition at line 1106 of file [trac.c](#).

```

01111     {
01112
01113     /* Set timer... */
01114     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01115
01116     const int np = atm->np;
01117     #ifdef _OPENACC
01118     #pragma acc data present(ctl, met0, met1, atm, cache)
01119     #pragma acc parallel loop independent gang vector
01120     #else
01121     #pragma omp parallel for default(shared)
01122     #endif
01123     for (int ip = 0; ip < np; ip++) {
01124
01125         double t;
01126
01127         /* Init... */
01128         INTPOL_INIT;
01129
01130         /* Restore pressure... */
01131         if (ctl->isosurf == 1)
01132             atm->p[ip] = cache->iso_var[ip];
01133
01134         /* Restore density... */
01135         else if (ctl->isosurf == 2) {
01136             INTPOL_3D(t, 1);
01137             atm->p[ip] = cache->iso_var[ip] * t;
01138         }
01139
01140         /* Restore potential temperature... */
01141         else if (ctl->isosurf == 3) {
01142             INTPOL_3D(t, 1);
01143             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
01144         }
01145
01146         /* Interpolate pressure... */
01147         else if (ctl->isosurf == 4) {
01148             if (atm->time[ip] <= cache->iso_ts[0])
01149                 atm->p[ip] = cache->iso_ps[0];
01150             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
01151                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
01152             else {
01153                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
01154                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
01155                                cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
01156                                atm->time[ip]);
01157             }
01158         }
01159     }
01160 }

```

Here is the call graph for this function:



#### 5.45.2.10 module\_meteo() void module\_meteo (

```

    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm )

```

Interpolate meteo data for air parcel positions.

Definition at line 1164 of file [trac.c](#).

```

01169     {
01170
01171     /* Set timer... */
01172     SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);
01173
01174     /* Check quantity flags... */
01175     if (ctl->qnt_tsts >= 0)
01176         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01177             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01178
01179     const int np = atm->np;
01180 #ifdef _OPENACC
01181 #pragma acc data present(ctl, clim, met0, met1, atm)
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 < np; ip++) {
01187
01188         double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
01189             cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
01190
01191         /* Interpolate meteo data... */
01192         INTPOL_INIT;
01193         INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
01194
01195         /* Set quantities... */
01196         SET_ATM(qnt_ps, ps);
01197         SET_ATM(qnt_ts, ts);
01198         SET_ATM(qnt_zs, zs);
01199         SET_ATM(qnt_us, us);
01200         SET_ATM(qnt_vs, vs);
01201         SET_ATM(qnt_pbl, pbl);
01202         SET_ATM(qnt_pt, pt);
01203         SET_ATM(qnt_tt, tt);
01204         SET_ATM(qnt_zt, zt);
01205         SET_ATM(qnt_h2ot, h2ot);
01206         SET_ATM(qnt_z, z);
01207         SET_ATM(qnt_p, atm->p[ip]);
01208         SET_ATM(qnt_t, t);
01209         SET_ATM(qnt_rho, RHO(atm->p[ip], t));
01210         SET_ATM(qnt_u, u);
01211         SET_ATM(qnt_v, v);
01212         SET_ATM(qnt_w, w);
01213         SET_ATM(qnt_h2o, h2o);
01214         SET_ATM(qnt_o3, o3);
01215         SET_ATM(qnt_lwc, lwc);
01216         SET_ATM(qnt_iwc, iwc);

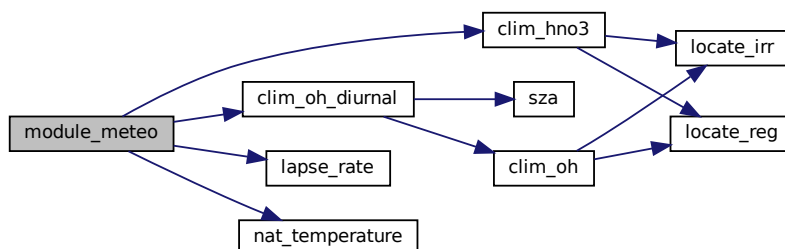
```

```

01217     SET_ATM(qnt_pct, pct);
01218     SET_ATM(qnt_pcb, pcb);
01219     SET_ATM(qnt_cl, cl);
01220     SET_ATM(qnt_plcl, plcl);
01221     SET_ATM(qnt_plfc, plfc);
01222     SET_ATM(qnt_pel, pel);
01223     SET_ATM(qnt_cape, cape);
01224     SET_ATM(qnt_cin, cin);
01225     SET_ATM(qnt_hno3,
01226             clim_hno3(clim, atm->time[ip], atm->lat[ip], atm->p[ip]));
01227     SET_ATM(qnt_oh,
01228             clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01229                             atm->lon[ip], atm->lat[ip]));
01230     SET_ATM(qnt_vh, sqrt(u * u + v * v));
01231     SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01232     SET_ATM(qnt_psat, PSAT(t));
01233     SET_ATM(qnt_psice, PSICE(t));
01234     SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
01235     SET_ATM(qnt_sh, SH(h2o));
01236     SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
01237     SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01238     SET_ATM(qnt_theta, THETA(atm->p[ip], t));
01239     SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
01240     SET_ATM(qnt_tvirt, TVIRT(t, h2o));
01241     SET_ATM(qnt_lapse, lapse_rate(t, h2o));
01242     SET_ATM(qnt_pv, pv);
01243     SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
01244     SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
01245     SET_ATM(qnt_tnat,
01246             nat_temperature(atm->p[ip], h2o,
01247                             clim_hno3(clim, atm->time[ip], atm->lat[ip],
01248                                     atm->p[ip])));
01249     SET_ATM(qnt_tsts,
01250             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01251 }
01252 }

```

Here is the call graph for this function:



```

5.45.2.11 module_oh_chem() void module_oh_chem (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate OH chemistry.

Definition at line 1256 of file [trac.c](#).

```

01262 {
01263
01264     /* Set timer... */

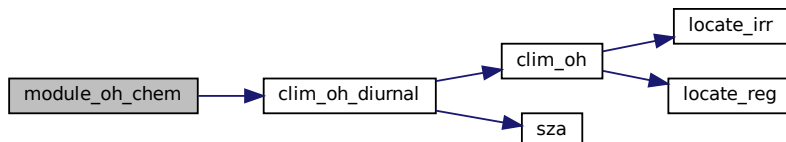
```

```

01265     SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);
01266
01267     /* Check quantity flags... */
01268     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01269         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01270
01271     const int np = atm->np;
01272     #ifdef _OPENACC
01273     #pragma acc data present(ctl, clim, met0, met1, atm, dt)
01274     #pragma acc parallel loop independent gang vector
01275     #else
01276     #pragma omp parallel for default(shared)
01277     #endif
01278     for (int ip = 0; ip < np; ip++)
01279         if (dt[ip] != 0) {
01280
01281             /* Get temperature... */
01282             double t;
01283             INTPOL_INIT;
01284             INTPOL_3D(t, 1);
01285
01286             /* Use constant reaction rate... */
01287             double k = GSL_NAN;
01288             if (ctl->oh_chem_reaction == 1)
01289                 k = ctl->oh_chem[0];
01290
01291             /* Calculate bimolecular reaction rate... */
01292             else if (ctl->oh_chem_reaction == 2)
01293                 k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);
01294
01295             /* Calculate termolecular reaction rate... */
01296             if (ctl->oh_chem_reaction == 3) {
01297
01298                 /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
01299                 double M = 7.243e21 * (atm->p[ip] / 1000.) / t;
01300
01301                 /* Calculate rate coefficient for X + OH + M -> XOH + M
01302                  (JPL Publication 19-05) ... */
01303                 double k0 = ctl->oh_chem[0] *
01304                     (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
01305                 double ki = ctl->oh_chem[2] *
01306                     (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
01307                 double c = log10(k0 * M / ki);
01308                 k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01309             }
01310
01311             /* Calculate exponential decay... */
01312             double rate_coef =
01313                 k * clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01314                                     atm->lon[ip],
01315                                     atm->lat[ip]);
01316             double aux = exp(-dt[ip] * rate_coef);
01317             if (ctl->qnt_m >= 0) {
01318                 if (ctl->qnt_mloss_oh >= 0)
01319                     atm->q[ctl->qnt_mloss_oh][ip]
01320                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01321                 atm->q[ctl->qnt_m][ip] *= aux;
01322             }
01323             if (ctl->qnt_vmr >= 0)
01324                 atm->q[ctl->qnt_vmr][ip] *= aux;
01325         }
01326 }

```

Here is the call graph for this function:



```

5.45.2.12 module_h2o2_chem() void module_h2o2_chem (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate H2O2 chemistry.

Definition at line 1330 of file [trac.c](#).

```

1337     {
1338
1339     /* Set timer... */
1340     SELECT_TIMER("MODULE_H2O2CHEM", "PHYSICS", NVTX_GPU);
1341
1342     /* Check quantity flags... */
1343     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
1344         ERRMSG("Module needs quantity mass or volume mixing ratio!");
1345     if (ctl->qnt_vmrimpl < 0)
1346         ERRMSG("Module needs quantity implicit volume mixing ratio!");
1347
1348     /* Create random numbers... */
1349     module_rng(rs, (size_t) atm->np, 0);
1350
1351     const int np = atm->np;
1352 #ifdef _OPENACC
1353 #pragma acc data present(clim,ctl,met0,met1,atm,dt,rs)
1354 #pragma acc parallel loop independent gang vector
1355 #else
1356 #pragma omp parallel for default(shared)
1357 #endif
1358     for (int ip = 0; ip < np; ip++)
1359         if (dt[ip] != 0) {
1360
1361             /* Check whether particle is inside cloud... */
1362             double lwc, iwc;
1363             INTPOL_INIT;
1364             INTPOL_3D(lwc, 1);
1365             INTPOL_3D(iwc, 0);
1366             if (!(lwc > 0 || iwc > 0))
1367                 continue;
1368
1369             /* Check cloud cover... */
1370             if (rs[ip] > ctl->h2o2_chem_cc)
1371                 continue;
1372
1373             /* Check implicit volume mixing ratio... */
1374             if (atm->q[ctl->qnt_vmrimpl][ip] == 0)
1375                 continue;
1376
1377             /* Get temperature... */
1378             double t;
1379             INTPOL_3D(t, 0);
1380
1381             /* Reaction rate (Berglen et al., 2004)... */
1382             double k = 9.1e7 * exp(-29700 / RI * (1. / t - 1. / 298.15)); // Maass 1999 unit: M^(-2)
1383
1384             /* Henry constant of SO2... */
1385             double H_SO2 = 1.3e-2 * exp(2900 * (1. / t - 1. / 298.15)) * RI * t;
1386             double K_1S = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15)); // unit: M
1387
1388             /* Henry constant of H2O2... */
1389             double H_h2o2 = 8.3e2 * exp(7600 * (1 / t - 1 / 298.15)) * RI * t;
1390
1391             /* Concentration of H2O2 (Barth et al., 1989)... */
1392             double SO2 = atm->q[ctl->qnt_vmrimpl][ip] * 1e9; // vmr unit: ppbv
1393             double h2o2 = H_h2o2
1394                 * clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip])
1395                 * 0.59 * exp(-0.687 * SO2) * 1000 / 6.02214e23; // unit: M
1396
1397             /* Volume water content in cloud [m^3 m^(-3)]... */
1398             double rho_air = 100 * atm->p[ip] / (RA * t);
1399             double CWC = lwc * rho_air / 1000 + iwc * rho_air / 920;
1400
1401             /* Calculate exponential decay (Rolph et al., 1992)... */
1402             double rate_coef = k * K_1S * h2o2 * H_SO2 * CWC;
1403             double aux = exp(-dt[ip] * rate_coef);
1404             if (ctl->qnt_m >= 0) {
1405                 if (ctl->qnt_mloss_h2o2 >= 0)
1406                     atm->q[ctl->qnt_mloss_h2o2][ip] +=

```

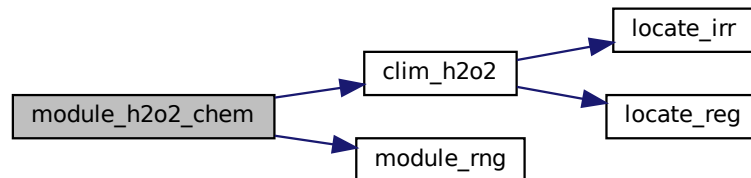


```

01407         atm->q[ctl->qnt_m][ip] * (1 - aux);
01408     atm->q[ctl->qnt_m][ip] *= aux;
01409 }
01410 if (ctl->qnt_vmr >= 0)
01411     atm->q[ctl->qnt_vmr][ip] *= aux;
01412 }
01413 }

```

Here is the call graph for this function:



**5.45.2.13 module\_position()** void module\_position (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Check position of air parcels.

Definition at line 1417 of file `trac.c`.

```

01422     {
01423
01424         /* Set timer... */
01425         SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);
01426
01427         const int np = atm->np;
01428 #ifdef _OPENACC
01429 #pragma acc data present(met0, met1, atm, dt)
01430 #pragma acc parallel loop independent gang vector
01431 #else
01432 #pragma omp parallel for default(shared)
01433 #endif
01434         for (int ip = 0; ip < np; ip++)
01435             if (dt[ip] != 0) {
01436
01437                 /* Init... */
01438                 double ps;
01439                 INTPOL_INIT;
01440
01441                 /* Calculate modulo... */
01442                 atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01443                 atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01444
01445                 /* Check latitude... */
01446                 while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01447                     if (atm->lat[ip] > 90) {
01448                         atm->lat[ip] = 180 - atm->lat[ip];
01449                         atm->lon[ip] += 180;
01450                     }
01451                     if (atm->lat[ip] < -90) {
01452                         atm->lat[ip] = -180 - atm->lat[ip];
01453                         atm->lon[ip] += 180;
01454                     }
01455                 }
01456             }

```

```

01457      /* Check longitude... */
01458      while (atm->lon[ip] < -180)
01459          atm->lon[ip] += 360;
01460      while (atm->lon[ip] >= 180)
01461          atm->lon[ip] -= 360;
01462
01463      /* Check pressure... */
01464      if (atm->p[ip] < met0->p[met0->np - 1]) {
01465          if (ctl->reflect)
01466              atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
01467          else
01468              atm->p[ip] = met0->p[met0->np - 1];
01469      } else if (atm->p[ip] > 300.) {
01470          INTPOL_2D(ps, 1);
01471          if (atm->p[ip] > ps) {
01472              if (ctl->reflect)
01473                  atm->p[ip] = 2. * ps - atm->p[ip];
01474              else
01475                  atm->p[ip] = ps;
01476          }
01477      }
01478 }
01479 }

```

**5.45.2.14 module\_rng\_init()** void module\_rng\_init (  
int ntask )

Initialize random number generator...

Definition at line 1483 of file [trac.c](#).

```

01484      {
01485
01486      /* Initialize random number generator... */
01487      #ifdef _OPENACC
01488
01489          if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT) !=
01490              CURAND_STATUS_SUCCESS)
01491              ERRMSG("Cannot create random number generator!");
01492          if (curandSetPseudoRandomGeneratorSeed(rng, ntask) != CURAND_STATUS_SUCCESS)
01493              ERRMSG("Cannot set seed for random number generator!");
01494          if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01495              != CURAND_STATUS_SUCCESS)
01496              ERRMSG("Cannot set stream for random number generator!");
01497
01498      #else
01499
01500          gsl_rng_env_setup();
01501          if (omp_get_max_threads() > NTHREADS)
01502              ERRMSG("Too many threads!");
01503          for (int i = 0; i < NTHREADS; i++) {
01504              rng[i] = gsl_rng_alloc(gsl_rng_default);
01505              gsl_rng_set(rng[i],
01506                          gsl_rng_default_seed + (long unsigned) (ntask * NTHREADS +
01507                          i));
01508          }
01509
01510      #endif
01511 }

```

**5.45.2.15 module\_rng()** void module\_rng (  
double \* rs,  
size\_t n,  
int method )

Generate random numbers.

Definition at line 1515 of file [trac.c](#).

```

01518      {
01519
01520      #ifdef _OPENACC

```

```

01521
01522 #pragma acc host_data use_device(rs)
01523 {
01524     /* Uniform distribution... */
01525     if (method == 0) {
01526         if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n)) !=
01527             CURAND_STATUS_SUCCESS)
01528             ERRMSG("Cannot create random numbers!");
01529     }
01530
01531     /* Normal distribution... */
01532     else if (method == 1) {
01533         if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0) !=
01534             CURAND_STATUS_SUCCESS)
01535             ERRMSG("Cannot create random numbers!");
01536     }
01537 }
01538
01539 #else
01540
01541     /* Uniform distribution... */
01542     if (method == 0) {
01543 #pragma omp parallel for default(shared)
01544         for (size_t i = 0; i < n; ++i)
01545             rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01546     }
01547
01548     /* Normal distribution... */
01549     else if (method == 1) {
01550 #pragma omp parallel for default(shared)
01551         for (size_t i = 0; i < n; ++i)
01552             rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01553     }
01554 #endif
01555 }

```

**5.45.2.16 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 1559 of file [trac.c](#).

```

01564     {
01565
01566         /* Set timer... */
01567         SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);
01568
01569         const int np = atm->np;
01570 #ifdef _OPENACC
01571 #pragma acc data present(ctl, met0, met1, atm, dt)
01572 #pragma acc parallel loop independent gang vector
01573 #else
01574 #pragma omp parallel for default(shared)
01575 #endif
01576         for (int ip = 0; ip < np; ip++)
01577             if (dt[ip] != 0) {
01578
01579                 /* Get temperature... */
01580                 double t;
01581                 INTPOL_INIT;
01582                 INTPOL_3D(t, 1);
01583
01584                 /* Sedimentation velocity... */
01585                 double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01586                     atm->q[ctl->qnt_rhop][ip]);
01587
01588                 /* Calculate pressure change... */
01589                 atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01590             }
01591     }

```

Here is the call graph for this function:



**5.45.2.17 module\_sort()** void module\_sort (

```

    ctl_t * ctl,
    met_t * met0,
    atm_t * atm )

```

Sort particles according to box index.

Definition at line 1595 of file [trac.c](#).

```

01598     {
01599
01600     /* Set timer... */
01601     SELECT_TIMER("MODULE_SORT", "PHYSICS", NVTX_GPU);
01602
01603     /* Allocate... */
01604     const int np = atm->np;
01605     double *restrict const a = (double *) malloc((size_t) np * sizeof(double));
01606     int *restrict const p = (int *) malloc((size_t) np * sizeof(int));
01607
01608     #ifdef _OPENACC
01609     #pragma acc enter data create(a[0:np], p[0:np])
01610     #pragma acc data present(ctl, met0, atm, a, p)
01611     #endif
01612
01613     /* Get box index... */
01614     #ifdef _OPENACC
01615     #pragma acc parallel loop independent gang vector
01616     #else
01617     #pragma omp parallel for default(shared)
01618     #endif
01619     for (int ip = 0; ip < np; ip++) {
01620         a[ip] =
01621             (double) ((locate_reg(met0->lon, met0->nx, atm->lon[ip]) * met0->ny +
01622                 locate_reg(met0->lat, met0->ny,
01623                     atm->lat[ip])) * met0->np + locate_irr(met0->p,
01624                         met0->np,
01625                         atm->p
01626                         [ip]));
01627         p[ip] = ip;
01628     }
01629
01630     /* Sorting... */
01631     #ifdef _OPENACC
01632     {
01633     #ifdef THRUST
01634     {
01635     #pragma acc host_data use_device(a, p)
01636     thrustSortWrapper(a, np, p);
01637     }
01638     #else
01639     {
01640     #pragma acc update host(a[0:np], p[0:np])
01641     #pragma omp parallel
01642     {
01643     #pragma omp single nowait
01644     quicksort(a, p, 0, np - 1);
01645     }
01646     #pragma acc update device(a[0:np], p[0:np])
01647     }
01648     #endif

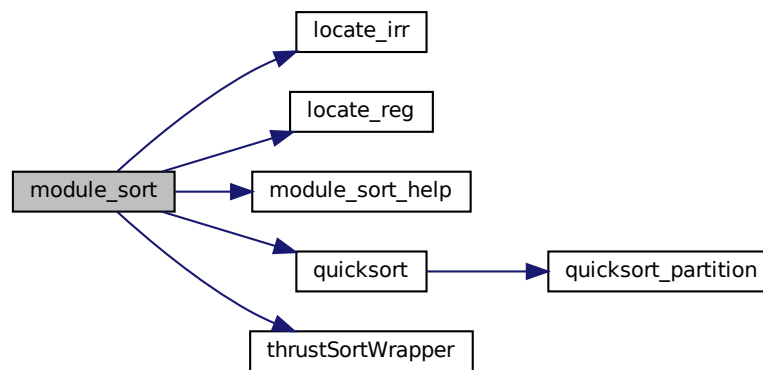
```

```

01649     }
01650     #else
01651     {
01652     #pragma omp parallel
01653     {
01654     #pragma omp single nowait
01655         quicksort(a, p, 0, np - 1);
01656     }
01657     }
01658     #endif
01659
01660     /* Sort data... */
01661     module_sort_help(atm->time, p, np);
01662     module_sort_help(atm->p, p, np);
01663     module_sort_help(atm->lon, p, np);
01664     module_sort_help(atm->lat, p, np);
01665     for (int iq = 0; iq < ctl->nq; iq++)
01666         module_sort_help(atm->q[iq], p, np);
01667
01668     /* Free... */
01669     #ifdef _OPENACC
01700 #pragma acc exit data delete(a,p)
01701 #endif
01702     free(a);
01703     free(p);
01704 }

```

Here is the call graph for this function:



**5.45.2.18 module\_sort\_help()** void module\_sort\_help (

```

double * a,
int * p,
int np )

```

Helper function for sorting module.

Definition at line 1678 of file [trac.c](#).

```

01681     {
01682
01683     /* Allocate... */
01684     double *restrict const help =
01685         (double *) malloc((size_t) np * sizeof(double));
01686
01687     /* Reordering of array... */
01688     #ifdef _OPENACC
01689     #pragma acc enter data create(help[0:np])
01690     #pragma acc data present(a,p,help)
01691     #pragma acc parallel loop independent gang vector

```

```

01692 #endif
01693     for (int ip = 0; ip < np; ip++)
01694         help[ip] = a[p[ip]];
01695 #ifdef _OPENACC
01696 #pragma acc parallel loop independent gang vector
01697 #endif
01698     for (int ip = 0; ip < np; ip++)
01699         a[ip] = help[ip];
01700
01701     /* Free... */
01702 #ifdef _OPENACC
01703 #pragma acc exit data delete(help)
01704 #endif
01705     free(help);
01706 }

```

**5.45.2.19 module\_timesteps()** void module\_timesteps (

```

    ctl_t * ctl,
    atm_t * atm,
    double * dt,
    double t )

```

Calculate time steps.

Definition at line 1710 of file [trac.c](#).

```

01714     {
01715
01716     /* Set timer... */
01717     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01718
01719     const int np = atm->np;
01720 #ifdef _OPENACC
01721 #pragma acc data present(ctl, atm, dt)
01722 #pragma acc parallel loop independent gang vector
01723 #else
01724 #pragma omp parallel for default(shared)
01725 #endif
01726     for (int ip = 0; ip < np; ip++) {
01727         if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
01728             && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
01729             && ctl->direction * (atm->time[ip] - t) < 0))
01730             dt[ip] = t - atm->time[ip];
01731         else
01732             dt[ip] = 0.0;
01733     }
01734 }

```

**5.45.2.20 module\_timesteps\_init()** void module\_timesteps\_init (

```

    ctl_t * ctl,
    atm_t * atm )

```

Initialize timesteps.

Definition at line 1738 of file [trac.c](#).

```

01740     {
01741
01742     /* Set timer... */
01743     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01744
01745     /* Set start time... */
01746     if (ctl->direction == 1) {
01747         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01748         if (ctl->t_stop > 1e99)
01749             ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01750     } else {
01751         ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01752         if (ctl->t_stop > 1e99)
01753             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01754     }

```

```

01755
01756  /* Check time interval... */
01757  if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
01758      ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
01759
01760  /* Round start time... */
01761  if (ctl->direction == 1)
01762      ctl->t_start = floor(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01763  else
01764      ctl->t_start = ceil(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01765 }

```

**5.45.2.21 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 1769 of file [trac.c](#).

```

01774  {
01775
01776  /* Set timer... */
01777  SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);
01778
01779  /* Check quantity flags... */
01780  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01781      ERRMSG("Module needs quantity mass or volume mixing ratio!");
01782
01783  const int np = atm->np;
01784  #ifdef _OPENACC
01785  #pragma acc data present(ctl, met0, met1, atm, dt)
01786  #pragma acc parallel loop independent gang vector
01787  #else
01788  #pragma omp parallel for default(shared)
01789  #endif
01790  for (int ip = 0; ip < np; ip++)
01791      if (dt[ip] != 0) {
01792
01793          double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;
01794
01795          /* Check whether particle is below cloud top... */
01796          INTPOL_INIT;
01797          INTPOL_2D(pct, 1);
01798          if (!isfinite(pct) || atm->p[ip] <= pct)
01799              continue;
01800
01801          /* Get cloud bottom pressure... */
01802          INTPOL_2D(pcb, 0);
01803
01804          /* Estimate precipitation rate (Pisso et al., 2019)... */
01805          INTPOL_2D(cl, 0);
01806          double Is =
01807              pow(1. / ctl->wet_depo_pre[0] * cl, 1. / ctl->wet_depo_pre[1]);
01808          if (Is < 0.01)
01809              continue;
01810
01811          /* Check whether particle is inside or below cloud... */
01812          INTPOL_3D(lwc, 1);
01813          INTPOL_3D(iwc, 0);
01814          int inside = (iwc > 0 || lwc > 0);
01815
01816          /* Get temperature... */
01817          INTPOL_3D(t, 0);
01818
01819          /* Calculate in-cloud scavenging coefficient... */
01820          if (inside) {
01821
01822              /* Calculate retention factor... */
01823              double eta;
01824              if (t > 273.15)
01825                  eta = 1;
01826              else if (t <= 238.15)
01827                  eta = ctl->wet_depo_ic_ret_ratio;
01828              else

```

```

01829     eta = LIN(273.15, 1, 238.15, ctl->wet_depo_ic_ret_ratio, t);
01830
01831     /* Use exponential dependency for particles ... */
01832     if (ctl->wet_depo_ic_a > 0)
01833         lambda = ctl->wet_depo_ic_a * pow(Is, ctl->wet_depo_ic_b) * eta;
01834
01835     /* Use Henry's law for gases... */
01836     else if (ctl->wet_depo_ic_h[0] > 0) {
01837
01838         /* Get Henry's constant (Sander, 2015)... */
01839         h = ctl->wet_depo_ic_h[0]
01840             * exp(ctl->wet_depo_ic_h[1] * (1. / t - 1. / 298.15));
01841
01842         /* Use effective Henry's constant for SO2
01843            (Berglen, 2004; Simpson, 2012)... */
01844         if (ctl->wet_depo_ic_h[2] > 0) {
01845             double H_ion = pow(10, ctl->wet_depo_ic_h[2] * (-1));
01846             double K_1 = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));
01847             double K_2 = 6e-8 * exp(1.12e3 * (1. / t - 1. / 298.15));
01848             h *= (1 + K_1 / H_ion + K_1 * K_2 / pow(H_ion, 2));
01849         }
01850
01851         /* Estimate depth of cloud layer... */
01852         dz = 1e3 * (Z(pct) - Z(pcb));
01853
01854         /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01855         lambda = h * RI * t * Is / 3.6e6 / dz * eta;
01856     }
01857 }
01858
01859 /* Calculate below-cloud scavenging coefficient... */
01860 else {
01861
01862     /* Calculate retention factor... */
01863     double eta;
01864     if (t > 270)
01865         eta = 1;
01866     else
01867         eta = ctl->wet_depo_bc_ret_ratio;
01868
01869     /* Use exponential dependency for particles... */
01870     if (ctl->wet_depo_bc_a > 0)
01871         lambda = ctl->wet_depo_bc_a * pow(Is, ctl->wet_depo_bc_b) * eta;
01872
01873     /* Use Henry's law for gases... */
01874     else if (ctl->wet_depo_bc_h[0] > 0) {
01875
01876         /* Get Henry's constant (Sander, 2015)... */
01877         h = ctl->wet_depo_bc_h[0]
01878             * exp(ctl->wet_depo_bc_h[1] * (1. / t - 1. / 298.15));
01879
01880         /* Estimate depth of cloud layer... */
01881         dz = 1e3 * (Z(pct) - Z(pcb));
01882
01883         /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01884         lambda = h * RI * t * Is / 3.6e6 / dz * eta;
01885     }
01886 }
01887
01888 /* Calculate exponential decay of mass... */
01889 double aux = exp(-dt[ip] * lambda);
01890 if (ctl->qnt_m >= 0) {
01891     if (ctl->qnt_mloss_wet >= 0)
01892         atm->q[ctl->qnt_mloss_wet][ip]
01893             += atm->q[ctl->qnt_m][ip] * (1 - aux);
01894     atm->q[ctl->qnt_m][ip] *= aux;
01895 }
01896 if (ctl->qnt_vmr >= 0)
01897     atm->q[ctl->qnt_vmr][ip] *= aux;
01898 }
01899 }

```

```

5.45.2.22 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.

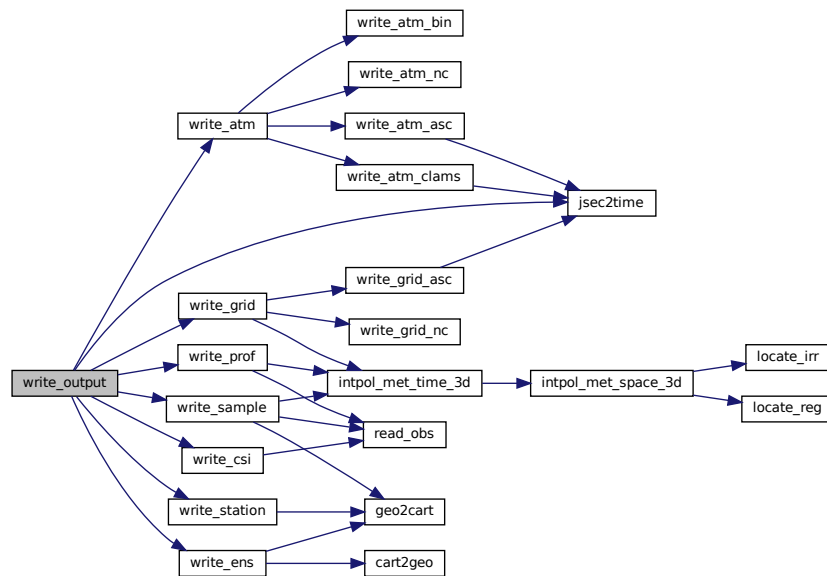
Definition at line 1903 of file `trac.c`.

```

01909     {
01910
01911     char ext[10], filename[2 * LEN];
01912
01913     double r;
01914
01915     int year, mon, day, hour, min, sec;
01916
01917     /* Get time... */
01918     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01919
01920     /* Update host... */
01921     #ifndef _OPENACC
01922     if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
01923         || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
01924         || (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0)
01925         || (ctl->csi_basename[0] != '-' || ctl->prof_basename[0] != '-'
01926             || ctl->sample_basename[0] != '-' || ctl->stat_basename[0] != '-') {
01927         SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
01928         #pragma acc update host(atm[:1])
01929     }
01930     #endif
01931
01932     /* Write atmospheric data... */
01933     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01934         if (ctl->atm_type == 0)
01935             sprintf(ext, "tab");
01936         else if (ctl->atm_type == 1)
01937             sprintf(ext, "bin");
01938         else if (ctl->atm_type == 2)
01939             sprintf(ext, "nc");
01940         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
01941             dirname, ctl->atm_basename, year, mon, day, hour, min, ext);
01942         write_atm(filename, ctl, atm, t);
01943     }
01944
01945     /* Write gridded data... */
01946     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01947         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
01948             dirname, ctl->grid_basename, year, mon, day, hour, min,
01949             ctl->grid_type == 0 ? "tab" : "nc");
01950         write_grid(filename, ctl, met0, met1, atm, t);
01951     }
01952
01953     /* Write CSI data... */
01954     if (ctl->csi_basename[0] != '-') {
01955         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01956         write_csi(filename, ctl, atm, t);
01957     }
01958
01959     /* Write ensemble data... */
01960     if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
01961         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01962             dirname, ctl->ens_basename, year, mon, day, hour, min);
01963         write_ens(filename, ctl, atm, t);
01964     }
01965
01966     /* Write profile data... */
01967     if (ctl->prof_basename[0] != '-') {
01968         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01969         write_prof(filename, ctl, met0, met1, atm, t);
01970     }
01971
01972     /* Write sample data... */
01973     if (ctl->sample_basename[0] != '-') {
01974         sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
01975         write_sample(filename, ctl, met0, met1, atm, t);
01976     }
01977
01978     /* Write station data... */
01979     if (ctl->stat_basename[0] != '-') {
01980         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01981         write_station(filename, ctl, atm, t);
01982     }
01983 }

```

Here is the call graph for this function:



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

Definition at line 213 of file [trac.c](#).

```

00215     {
00216
00217     ctl_t ctl;
00218
00219     atm_t *atm;
00220
00221     cache_t *cache;
00222
00223     clim_t *clim;
00224
00225     met_t *met0, *met1;
00226
00227 #ifdef ASYNCIO
00228     met_t *met0TMP, *met1TMP, *mets;
00229     ctl_t ctlTMP;
00230 #endif
00231
00232     FILE *dirlist;
00233
00234     char dirname[LEN], filename[2 * LEN];
00235
00236     double *dt, *rs, t;
00237
00238     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00239
00240     /* Start timers... */
00241     START_TIMERS;
00242
00243     /* Initialize MPI... */
00244 #ifdef MPI
00245     SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
00246     MPI_Init(&argc, &argv);
00247     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00248     MPI_Comm_size(MPI_COMM_WORLD, &size);
00249 #endif
00250

```

```

00251  /* Initialize GPUs... */
00252  #ifdef _OPENACC
00253      SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
00254      num_devices = acc_get_num_devices(acc_device_nvidia);
00255      if (num_devices <= 0)
00256          ERRMSG("Not running on a GPU device!");
00257      int device_num = rank % num_devices;
00258      acc_set_device_num(device_num, acc_device_nvidia);
00259      acc_device_t device_type = acc_get_device_type();
00260      acc_init(device_type);
00261  #endif
00262
00263  /* Check arguments... */
00264  if (argc < 4)
00265      ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00266
00267  /* Open directory list... */
00268  if (!(dirlist = fopen(argv[1], "r")))
00269      ERRMSG("Cannot open directory list!");
00270
00271  /* Loop over directories... */
00272  while (fscanf(dirlist, "%4999s", dirname) != EOF) {
00273
00274      /* MPI parallelization... */
00275      if ((++ntask) % size != rank)
00276          continue;
00277
00278      /* -----
00279      Initialize model run...
00280      ----- */
00281
00282      /* Allocate... */
00283      SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
00284      ALLOC(atm, atm_t, 1);
00285      ALLOC(cache, cache_t, 1);
00286      ALLOC(clim, clim_t, 1);
00287      ALLOC(met0, met_t, 1);
00288      ALLOC(met1, met_t, 1);
00289  #ifdef ASYNCIO
00290      ALLOC(met0TMP, met_t, 1);
00291      ALLOC(met1TMP, met_t, 1);
00292  #endif
00293      ALLOC(dt, double,
00294            NP);
00295      ALLOC(rs, double,
00296            3 * NP + 1);
00297
00298      /* Create data region on GPUs... */
00299  #ifdef _OPENACC
00300      SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
00301  #ifdef ASYNCIO
00302      #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl,ctlTMP, met0[:1], met1[:1],
00303                                  met0TMP[:1], met1TMP[:1], dt[:NP], rs[:3 * NP])
00304  #else
00305      #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3 *
00306                                  NP])
00307  #endif
00308      #endif
00309
00310      /* Read control parameters... */
00311      sprintf(filename, "%s/%s", dirname, argv[2]);
00312      read_ctl(filename, argc, argv, &ctl);
00313
00314      /* Read climatological data... */
00315      read_clim(&ctl, clim);
00316
00317      /* Read atmospheric data... */
00318      sprintf(filename, "%s/%s", dirname, argv[3]);
00319      if (!read_atm(filename, &ctl, atm))
00320          ERRMSG("Cannot open file!");
00321
00322      /* Initialize timesteps... */
00323      module_timesteps_init(&ctl, atm);
00324
00325      /* Update GPU... */
00326  #ifdef _OPENACC
00327      SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00328      #pragma acc update device(atm[:1], clim[:1], ctl)
00329  #endif
00330
00331      /* Initialize random number generator... */
00332      module_rng_init(ntask);
00333
00334      /* Initialize meteo data... */
00335  #ifdef ASYNCIO
00336      ctlTMP = ctl;
00337  #endif

```

```

00336     get_met(&ctl, clim, ctl.t_start, &met0, &met1);
00337     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00338         WARN("Violation of CFL criterion! Check DT_MOD!");
00339 #ifdef ASYNCIO
00340     get_met(&ctlTMP, clim, ctlTMP.t_start, &met0TMP, &met1TMP);
00341 #endif
00342
00343     /* Initialize isosurface... */
00344     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00345         module_isosurf_init(&ctl, met0, met1, atm, cache);
00346
00347     /* Update GPU... */
00348 #ifdef _OPENACC
00349     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00350 #pragma acc update device(cache[:1])
00351 #endif
00352
00353     /* -----
00354     Loop over timesteps...
00355     ----- */
00356
00357     /* Loop over timesteps... */
00358 #ifdef ASYNCIO
00359     omp_set_nested(1);
00360     // omp_set_dynamic(0);
00361     int ompTrdnun = omp_get_max_threads();
00362 #endif
00363     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00364          t += ctl.direction * ctl.dt_mod) {
00365 #ifdef ASYNCIO
00366 #pragma omp parallel num_threads(2)
00367 {
00368 #endif
00369
00370         /* Adjust length of final time step... */
00371         if (ctl.direction * (t - ctl.t_stop) > 0)
00372             t = ctl.t_stop;
00373
00374         /* Set time steps of air parcels... */
00375         module_timesteps(&ctl, atm, dt, t);
00376
00377         /* Get meteo data... */
00378 #ifdef ASYNCIO
00379 #pragma acc wait(5)
00380 #pragma omp barrier
00381         if (omp_get_thread_num() == 0) {
00382
00383             /* Pointer swap... */
00384             if (t != ctl.t_start) {
00385                 mets = met0;
00386                 met0 = met0TMP;
00387                 met0TMP = mets;
00388
00389                 mets = met1;
00390                 met1 = met1TMP;
00391                 met1TMP = mets;
00392             }
00393 #endif
00394 #ifndef ASYNCIO
00395         if (t != ctl.t_start)
00396             get_met(&ctl, clim, t, &met0, &met1);
00397 #endif
00398
00399         /* Sort particles... */
00400         if (ctl.sort_dt > 0 && fmod(t, ctl.sort_dt) == 0)
00401             module_sort(&ctl, met0, atm);
00402
00403         /* Check initial positions... */
00404         module_position(&ctl, met0, met1, atm, dt);
00405
00406         /* Advection... */
00407         module_advect(&ctl, met0, met1, atm, dt);
00408
00409         /* Turbulent diffusion... */
00410         if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00411             || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
00412             module_diffusion_turb(&ctl, clim, atm, dt, rs);
00413
00414         /* Mesoscale diffusion... */
00415         if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
00416             module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00417
00418         /* Convection... */
00419         if (ctl.conv_cape >= 0
00420             && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
00421             module_convection(&ctl, met0, met1, atm, dt, rs);
00422

```

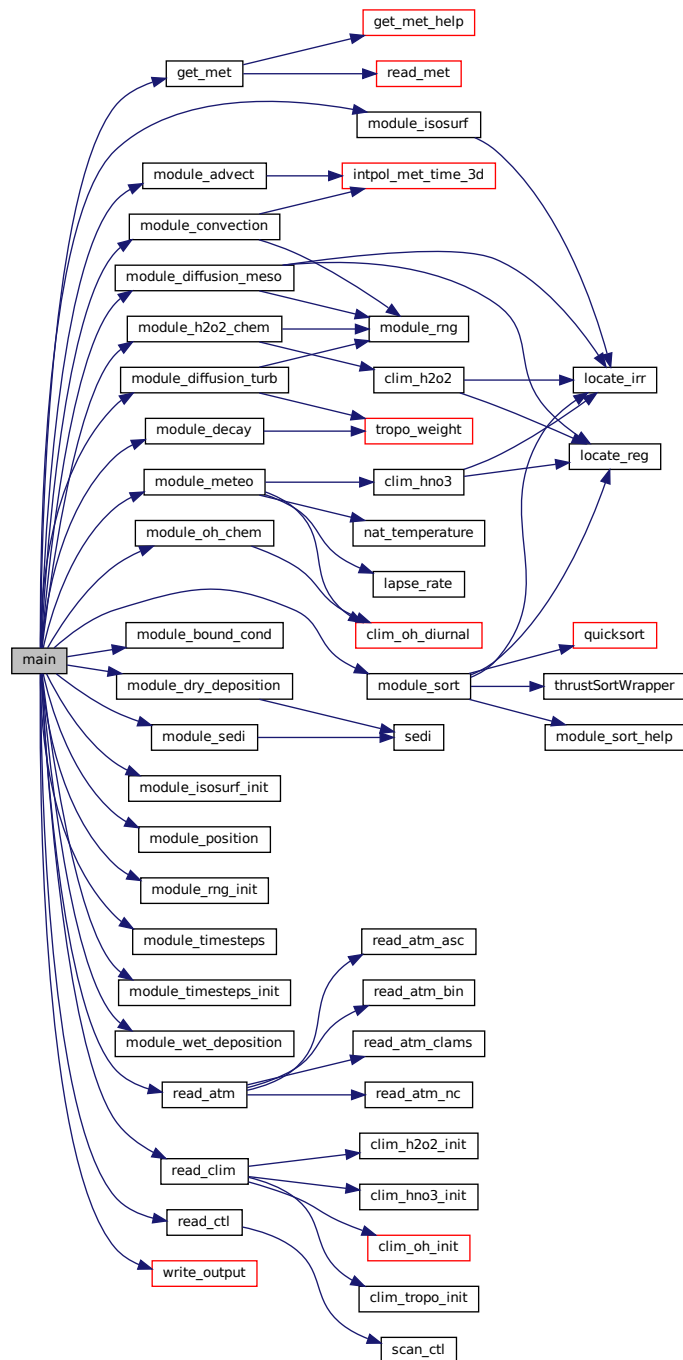
```

00423      /* Sedimentation... */
00424      if (ctl.qnt_rp >= 0 && ctl.qnt_rhop >= 0)
00425          module_sedi(&ctl, met0, met1, atm, dt);
00426
00427      /* Isosurface... */
00428      if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00429          module_isosurf(&ctl, met0, met1, atm, cache);
00430
00431      /* Check final positions... */
00432      module_position(&ctl, met0, met1, atm, dt);
00433
00434      /* Interpolate meteo data... */
00435      if (ctl.met_dt_out > 0
00436          && (ctl.met_dt_out < ctl.dt_mod
00437              || fmod(t, ctl.met_dt_out) == 0))
00438          module_meteo(&ctl, clim, met0, met1, atm);
00439
00440      /* Decay of particle mass... */
00441      if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00442          module_decay(&ctl, clim, atm, dt);
00443
00444      /* OH chemistry... */
00445      if (ctl.clim_oh_filename[0] != '-' && ctl.oh_chem_reaction != 0)
00446          module_oh_chem(&ctl, clim, met0, met1, atm, dt);
00447
00448      /* H2O2 chemistry (for SO2 aqueous phase oxidation)... */
00449      if (ctl.clim_h2o2_filename[0] != '-' && ctl.h2o2_chem_reaction != 0)
00450          module_h2o2_chem(&ctl, clim, met0, met1, atm, dt, rs);
00451
00452      /* Dry deposition... */
00453      if (ctl.dry_depo[0] > 0)
00454          module_dry_deposition(&ctl, met0, met1, atm, dt);
00455
00456      /* Wet deposition... */
00457      if ((ctl.wet_depo_ic_a > 0 || ctl.wet_depo_ic_h[0] > 0)
00458          && (ctl.wet_depo_bc_a > 0 || ctl.wet_depo_bc_h[0] > 0))
00459          module_wet_deposition(&ctl, met0, met1, atm, dt);
00460
00461      /* Boundary conditions... */
00462      if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
00463          module_bound_cond(&ctl, met0, met1, atm, dt);
00464
00465      write_output(dirname, &ctl, met0, met1, atm, t);
00466      #ifndef ASYNCIO
00467      } else {
00468          omp_set_num_threads(ompTrdnm);
00469          if (ctl.direction * (t - ctl.t_stop + ctl.direction * ctl.dt_mod) <
00470              ctl.dt_mod)
00471              get_met(&ctl, clim, t + (ctl.direction * ctl.dt_mod), &met0TMP,
00472                      &met1TMP);
00473      }
00474      #endif
00475      }
00476      }
00477
00478      #ifndef ASYNCIO
00479          omp_set_num_threads(ompTrdnm);
00480      #endif
00481
00482      /* -----
00483      Finalize model run...
00484      ----- */
00485
00486      /* Report problem size... */
00487      LOG(1, "SIZE_NP = %d", atm->np);
00488      LOG(1, "SIZE_MPI_TASKS = %d", size);
00489      LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
00490      LOG(1, "SIZE_ACC_DEVICES = %d", num_devices);
00491
00492      /* Report memory usage... */
00493      LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
00494      LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
00495      LOG(1, "MEMORY_CLIM = %g MByte", sizeof(clim_t) / 1024. / 1024.);
00496      LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
00497      LOG(1, "MEMORY_DYNAMIC = %g MByte", (3 * NP * sizeof(int)
00498          + 4 * NP * sizeof(double)
00499          + EX * EY * EP * sizeof(float)) /
00500          1024. / 1024.);
00501      LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * EP * sizeof(float)) /
00502          1024. / 1024.);
00503
00504      /* Delete data region on GPUs... */
00505      #ifdef _OPENACC
00506          SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
00507      #endif
00508      #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs, met0TMP, met1TMP)
00509      #else

```

```
00510 #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs)
00511 #endif
00512 #endif
00513
00514     /* Free... */
00515     SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
00516     free(atm);
00517     free(cache);
00518     free(clim);
00519     free(met0);
00520     free(met1);
00521 #ifndef ASYNCIO
00522     free(met0TMP);
00523     free(met1TMP);
00524 #endif
00525     free(dt);
00526     free(rs);
00527
00528     /* Report timers... */
00529     PRINT_TIMERS;
00530 }
00531
00532 /* Finalize MPI... */
00533 #ifdef MPI
00534     MPI_Finalize();
00535 #endif
00536
00537 /* Stop timers... */
00538     STOP_TIMERS;
00539
00540     return EXIT_SUCCESS;
00541 }
```

Here is the call graph for this function:



## 5.46 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) 2013-2022 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_advect(
00043     ctl_t * ctl,
00044     met_t * met0,
00045     met_t * met1,
00046     atm_t * atm,
00047     double *dt);
00048
00050 void module_bound_cond(
00051     ctl_t * ctl,
00052     met_t * met0,
00053     met_t * met1,
00054     atm_t * atm,
00055     double *dt);
00056
00058 void module_convection(
00059     ctl_t * ctl,
00060     met_t * met0,
00061     met_t * met1,
00062     atm_t * atm,
00063     double *dt,
00064     double *rs);
00065
00067 void module_decay(
00068     ctl_t * ctl,
00069     clim_t * clim,
00070     atm_t * atm,
00071     double *dt);
00072
00074 void module_diffusion_meso(
00075     ctl_t * ctl,
00076     met_t * met0,
00077     met_t * met1,
00078     atm_t * atm,
00079     cache_t * cache,
00080     double *dt,
00081     double *rs);
00082
00084 void module_diffusion_turb(
00085     ctl_t * ctl,
00086     clim_t * clim,
00087     atm_t * atm,
00088     double *dt,
00089     double *rs);
00090
00092 void module_dry_deposition(
00093     ctl_t * ctl,
00094     met_t * met0,
00095     met_t * met1,
00096     atm_t * atm,
00097     double *dt);
00098
00100 void module_isosurf_init(
00101     ctl_t * ctl,
00102     met_t * met0,
00103     met_t * met1,
00104     atm_t * atm,
00105     cache_t * cache);
00106
00108 void module_isosurf(
00109     ctl_t * ctl,

```



```
00110     met_t * met0,
00111     met_t * met1,
00112     atm_t * atm,
00113     cache_t * cache);
00114
00116 void module_meteo(
00117     ctl_t * ctl,
00118     clim_t * clim,
00119     met_t * met0,
00120     met_t * met1,
00121     atm_t * atm);
00122
00124 void module_oh_chem(
00125     ctl_t * ctl,
00126     clim_t * clim,
00127     met_t * met0,
00128     met_t * met1,
00129     atm_t * atm,
00130     double *dt);
00131
00133 void module_h2o2_chem(
00134     ctl_t * ctl,
00135     clim_t * clim,
00136     met_t * met0,
00137     met_t * met1,
00138     atm_t * atm,
00139     double *dt,
00140     double *rs);
00141
00143 void module_position(
00144     ctl_t * ctl,
00145     met_t * met0,
00146     met_t * met1,
00147     atm_t * atm,
00148     double *dt);
00149
00151 void module_rng_init(
00152     int ntask);
00153
00155 void module_rng(
00156     double *rs,
00157     size_t n,
00158     int method);
00159
00161 void module_sedi(
00162     ctl_t * ctl,
00163     met_t * met0,
00164     met_t * met1,
00165     atm_t * atm,
00166     double *dt);
00167
00169 void module_sort(
00170     ctl_t * ctl,
00171     met_t * met0,
00172     atm_t * atm);
00173
00175 void module_sort_help(
00176     double *a,
00177     int *p,
00178     int np);
00179
00181 void module_timesteps(
00182     ctl_t * ctl,
00183     atm_t * atm,
00184     double *dt,
00185     double t);
00186
00188 void module_timesteps_init(
00189     ctl_t * ctl,
00190     atm_t * atm);
00191
00193 void module_wet_deposition(
00194     ctl_t * ctl,
00195     met_t * met0,
00196     met_t * met1,
00197     atm_t * atm,
00198     double *dt);
00199
00201 void write_output(
00202     const char *dirname,
00203     ctl_t * ctl,
00204     met_t * met0,
00205     met_t * met1,
00206     atm_t * atm,
00207     double t);
00208
00209 /* -----
```

```

00210     Main...
00211     ----- */
00212
00213 int main(
00214     int argc,
00215     char *argv[]) {
00216     ctl_t ctl;
00217
00218     atm_t *atm;
00219
00220     cache_t *cache;
00221
00222     clim_t *clim;
00223
00224     met_t *met0, *met1;
00225
00226 #ifdef ASYNCIO
00227     met_t *met0TMP, *met1TMP, *mets;
00228     ctl_t ctlTMP;
00229 #endif
00230
00231     FILE *dirlist;
00232
00233     char dirname[LEN], filename[2 * LEN];
00234
00235     double *dt, *rs, t;
00236
00237     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00238
00239     /* Start timers... */
00240     START_TIMERS;
00241
00242     /* Initialize MPI... */
00243 #ifdef MPI
00244     SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
00245     MPI_Init(&argc, &argv);
00246     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00247     MPI_Comm_size(MPI_COMM_WORLD, &size);
00248 #endif
00249
00250     /* Initialize GPUs... */
00251 #ifdef _OPENACC
00252     SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
00253     num_devices = acc_get_num_devices(acc_device_nvidia);
00254     if (num_devices <= 0)
00255         ERRMSG("Not running on a GPU device!");
00256     int device_num = rank % num_devices;
00257     acc_set_device_num(device_num, acc_device_nvidia);
00258     acc_device_t device_type = acc_get_device_type();
00259     acc_init(device_type);
00260 #endif
00261
00262     /* Check arguments... */
00263     if (argc < 4)
00264         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00265
00266     /* Open directory list... */
00267     if (!(dirlist = fopen(argv[1], "r")))
00268         ERRMSG("Cannot open directory list!");
00269
00270     /* Loop over directories... */
00271     while (fscanf(dirlist, "%4999s", dirname) != EOF) {
00272
00273         /* MPI parallelization... */
00274         if ((++ntask) % size != rank)
00275             continue;
00276
00277         /* -----
00278            Initialize model run...
00279            ----- */
00280
00281         /* Allocate... */
00282         SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
00283         ALLOC(atm, atm_t, 1);
00284         ALLOC(cache, cache_t, 1);
00285         ALLOC(clim, clim_t, 1);
00286         ALLOC(met0, met_t, 1);
00287         ALLOC(met1, met_t, 1);
00288 #ifdef ASYNCIO
00289         ALLOC(met0TMP, met_t, 1);
00290         ALLOC(met1TMP, met_t, 1);
00291 #endif
00292         ALLOC(dt, double,
00293             NP);
00294         ALLOC(rs, double,
00295             3 * NP + 1);
00296

```

```

00297
00298 /* Create data region on GPUs... */
00299 #ifdef _OPENACC
00300     SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
00301 #ifdef ASYNCIO
00302 #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl,ctlTMP, met0[:1], met1[:1],
00303                             met0TMP[:1], met1TMP[:1], dt[:NP], rs[:3 * NP])
00303 #else
00304 #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3 *
00305                             NP])
00305 #endif
00306 #endif
00307
00308 /* Read control parameters... */
00309 sprintf(filename, "%s/%s", dirname, argv[2]);
00310 read_ctl(filename, argc, argv, &ctl);
00311
00312 /* Read climatological data... */
00313 read_clim(&ctl, clim);
00314
00315 /* Read atmospheric data... */
00316 sprintf(filename, "%s/%s", dirname, argv[3]);
00317 if (!read_atm(filename, &ctl, atm))
00318     ERRMSG("Cannot open file!");
00319
00320 /* Initialize timesteps... */
00321 module_timesteps_init(&ctl, atm);
00322
00323 /* Update GPU... */
00324 #ifdef _OPENACC
00325     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00326 #pragma acc update device(atm[:1], clim[:1], ctl)
00327 #endif
00328
00329 /* Initialize random number generator... */
00330 module_rng_init(ntask);
00331
00332 /* Initialize meteo data... */
00333 #ifdef ASYNCIO
00334     ctlTMP = ctl;
00335 #endif
00336 get_met(&ctl, clim, ctl.t_start, &met0, &met1);
00337 if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00338     WARN("Violation of CFL criterion! Check DT_MOD!");
00339 #ifdef ASYNCIO
00340     get_met(&ctlTMP, clim, ctlTMP.t_start, &met0TMP, &met1TMP);
00341 #endif
00342
00343 /* Initialize isosurface... */
00344 if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00345     module_isosurf_init(&ctl, met0, met1, atm, cache);
00346
00347 /* Update GPU... */
00348 #ifdef _OPENACC
00349     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00350 #pragma acc update device(cache[:1])
00351 #endif
00352
00353 /* -----
00354     Loop over timesteps...
00355     ----- */
00356
00357 /* Loop over timesteps... */
00358 #ifdef ASYNCIO
00359     omp_set_nested(1);
00360     // omp_set_dynamic(0);
00361     int ompTrdnum = omp_get_max_threads();
00362 #endif
00363 for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00364      t += ctl.direction * ctl.dt_mod) {
00365     #ifdef ASYNCIO
00366     #pragma omp parallel num_threads(2)
00367     {
00368     #endif
00369
00370     /* Adjust length of final time step... */
00371     if (ctl.direction * (t - ctl.t_stop) > 0)
00372         t = ctl.t_stop;
00373
00374     /* Set time steps of air parcels... */
00375     module_timesteps(&ctl, atm, dt, t);
00376
00377     /* Get meteo data... */
00378     #ifdef ASYNCIO
00379     #pragma acc wait(5)
00380     #pragma omp barrier
00381     if (omp_get_thread_num() == 0) {

```

```

00382
00383     /* Pointer swap... */
00384     if (t != ctl.t_start) {
00385         mets = met0;
00386         met0 = met0TMP;
00387         met0TMP = mets;
00388
00389         mets = met1;
00390         met1 = met1TMP;
00391         met1TMP = mets;
00392     }
00393 #endif
00394 #ifndef ASYNCIO
00395     if (t != ctl.t_start)
00396         get_met(&ctl, clim, t, &met0, &met1);
00397 #endif
00398
00399     /* Sort particles... */
00400     if (ctl.sort_dt > 0 && fmod(t, ctl.sort_dt) == 0)
00401         module_sort(&ctl, met0, atm);
00402
00403     /* Check initial positions... */
00404     module_position(&ctl, met0, met1, atm, dt);
00405
00406     /* Advection... */
00407     module_advect(&ctl, met0, met1, atm, dt);
00408
00409     /* Turbulent diffusion... */
00410     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00411         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
00412         module_diffusion_turb(&ctl, clim, atm, dt, rs);
00413
00414     /* Mesoscale diffusion... */
00415     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
00416         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00417
00418     /* Convection... */
00419     if (ctl.conv_cape >= 0
00420         && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
00421         module_convection(&ctl, met0, met1, atm, dt, rs);
00422
00423     /* Sedimentation... */
00424     if (ctl.qnt_rp >= 0 && ctl.qnt_rhop >= 0)
00425         module_sedi(&ctl, met0, met1, atm, dt);
00426
00427     /* Isosurface... */
00428     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00429         module_isosurf(&ctl, met0, met1, atm, cache);
00430
00431     /* Check final positions... */
00432     module_position(&ctl, met0, met1, atm, dt);
00433
00434     /* Interpolate meteo data... */
00435     if (ctl.met_dt_out > 0
00436         && (ctl.met_dt_out < ctl.dt_mod
00437             || fmod(t, ctl.met_dt_out) == 0))
00438         module_meteo(&ctl, clim, met0, met1, atm);
00439
00440     /* Decay of particle mass... */
00441     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00442         module_decay(&ctl, clim, atm, dt);
00443
00444     /* OH chemistry... */
00445     if (ctl.clim_oh_filename[0] != '-' && ctl.oh_chem_reaction != 0)
00446         module_oh_chem(&ctl, clim, met0, met1, atm, dt);
00447
00448     /* H2O2 chemistry (for SO2 aqueous phase oxidation)... */
00449     if (ctl.clim_h2o2_filename[0] != '-' && ctl.h2o2_chem_reaction != 0)
00450         module_h2o2_chem(&ctl, clim, met0, met1, atm, dt, rs);
00451
00452     /* Dry deposition... */
00453     if (ctl.dry_depo[0] > 0)
00454         module_dry_deposition(&ctl, met0, met1, atm, dt);
00455
00456     /* Wet deposition... */
00457     if ((ctl.wet_depo_ic_a > 0 || ctl.wet_depo_ic_h[0] > 0)
00458         && (ctl.wet_depo_bc_a > 0 || ctl.wet_depo_bc_h[0] > 0))
00459         module_wet_deposition(&ctl, met0, met1, atm, dt);
00460
00461     /* Boundary conditions... */
00462     if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
00463         module_bound_cond(&ctl, met0, met1, atm, dt);
00464
00465     write_output(dirname, &ctl, met0, met1, atm, t);
00466 #ifdef ASYNCIO
00467     } else {
00468         omp_set_num_threads(ompTrdnum);

```

```

00469         if (ctl.direction * (t - ctl.t_stop + ctl.direction * ctl.dt_mod) <
00470             ctl.dt_mod)
00471             get_met(&ctl, clim, t + (ctl.direction * ctl.dt_mod), &met0TMP,
00472                   &met1TMP);
00473     }
00474 }
00475 #endif
00476 }
00477
00478 #ifdef ASYNCIO
00479     omp_set_num_threads(ompTrdnm);
00480 #endif
00481
00482     /* -----
00483     Finalize model run...
00484     ----- */
00485
00486     /* Report problem size... */
00487     LOG(1, "SIZE_NP = %d", atm->np);
00488     LOG(1, "SIZE_MPI_TASKS = %d", size);
00489     LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
00490     LOG(1, "SIZE_ACC_DEVICES = %d", num_devices);
00491
00492     /* Report memory usage... */
00493     LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
00494     LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
00495     LOG(1, "MEMORY_CLIM = %g MByte", sizeof(clim_t) / 1024. / 1024.);
00496     LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
00497     LOG(1, "MEMORY_DYNAMIC = %g MByte", (3 * NP * sizeof(int)
00498                                         + 4 * NP * sizeof(double)
00499                                         + EX * EY * EP * sizeof(float)) /
00500         1024. / 1024.);
00501     LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * EP * sizeof(float)) /
00502         1024. / 1024.);
00503
00504     /* Delete data region on GPUs... */
00505     #ifdef _OPENACC
00506         SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
00507     #ifdef ASYNCIO
00508         #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs, met0TMP, met1TMP)
00509     #else
00510         #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs)
00511     #endif
00512     #endif
00513
00514     /* Free... */
00515     SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
00516     free(atm);
00517     free(cache);
00518     free(clim);
00519     free(met0);
00520     free(met1);
00521     #ifdef ASYNCIO
00522         free(met0TMP);
00523         free(met1TMP);
00524     #endif
00525     free(dt);
00526     free(rs);
00527
00528     /* Report timers... */
00529     PRINT_TIMERS;
00530 }
00531
00532     /* Finalize MPI... */
00533     #ifdef MPI
00534         MPI_Finalize();
00535     #endif
00536
00537     /* Stop timers... */
00538     STOP_TIMERS;
00539
00540     return EXIT_SUCCESS;
00541 }
00542
00543     /*-----
00544     void module_advect(
00545         ctl_t * ctl,
00546         met_t * met0,
00547         met_t * met1,
00548         atm_t * atm,
00549         double *dt) {
00550
00551     /* Set timer... */
00552     SELECT_TIMER("MODULE_ADVECTION", "PHYSICS", NVTX_GPU);
00553
00554     const int np = atm->np;

```

```

00556 #ifdef _OPENACC
00557 #pragma acc data present(ctl,met0,met1,atm,dt)
00558 #pragma acc parallel loop independent gang vector
00559 #else
00560 #pragma omp parallel for default(shared)
00561 #endif
00562 for (int ip = 0; ip < np; ip++)
00563     if (dt[ip] != 0) {
00564
00565         /* Init... */
00566         double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];
00567
00568         /* Loop over integration nodes... */
00569         for (int i = 0; i < ctl->advect; i++) {
00570
00571             /* Set position... */
00572             if (i == 0) {
00573                 dts = 0.0;
00574                 x[0] = atm->lon[ip];
00575                 x[1] = atm->lat[ip];
00576                 x[2] = atm->p[ip];
00577             } else {
00578                 dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
00579                 x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
00580                 x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
00581                 x[2] = atm->p[ip] + dts * w[i - 1];
00582             }
00583             double tm = atm->time[ip] + dts;
00584
00585             /* Interpolate meteo data... */
00586 #ifdef UVW
00587             intpol_met_time_uvw(met0, met1, tm, x[2], x[0], x[1],
00588                               &u[i], &v[i], &w[i]);
00589 #else
00590             INTPOL_INIT;
00591             intpol_met_time_3d(met0, met0->u, met1, met1->u, tm,
00592                               x[2], x[0], x[1], &u[i], ci, cw, 1);
00593             intpol_met_time_3d(met0, met0->v, met1, met1->v, tm,
00594                               x[2], x[0], x[1], &v[i], ci, cw, 0);
00595             intpol_met_time_3d(met0, met0->w, met1, met1->w, tm,
00596                               x[2], x[0], x[1], &w[i], ci, cw, 0);
00597 #endif
00598
00599             /* Get mean wind... */
00600             double k = 1.0;
00601             if (ctl->advect == 2)
00602                 k = (i == 0 ? 0.0 : 1.0);
00603             else if (ctl->advect == 4)
00604                 k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
00605             um += k * u[i];
00606             vm += k * v[i];
00607             wm += k * w[i];
00608         }
00609
00610         /* Set new position... */
00611         atm->time[ip] += dt[ip];
00612         atm->lon[ip] += DX2DEG(dt[ip] * um / 1000.,
00613                               (ctl->advect == 2 ? x[1] : atm->lat[ip]));
00614         atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
00615         atm->p[ip] += dt[ip] * wm;
00616     }
00617 }
00618
00619 /*****
00620
00621 void module_bound_cond(
00622     ctl_t * ctl,
00623     met_t * met0,
00624     met_t * met1,
00625     atm_t * atm,
00626     double *dt) {
00627
00628     /* Set timer... */
00629     SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);
00630
00631     /* Check quantity flags... */
00632     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00633         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00634
00635     const int np = atm->np;
00636 #ifdef _OPENACC
00637 #pragma acc data present(ctl, met0, met1, atm, dt)
00638 #pragma acc parallel loop independent gang vector
00639 #else
00640 #pragma omp parallel for default(shared)
00641 #endif
00642     for (int ip = 0; ip < np; ip++)

```

```

00643     if (dt[ip] != 0) {
00644
00645         double ps;
00646
00647         /* Check latitude and pressure range... */
00648         if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
00649             || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
00650             continue;
00651
00652         /* Check surface layer... */
00653         if (ctl->bound_dps > 0) {
00654
00655             /* Get surface pressure... */
00656             INTPOL_INIT;
00657             INTPOL_2D(ps, 1);
00658
00659             /* Check whether particle is above the surface layer... */
00660             if (atm->p[ip] < ps - ctl->bound_dps)
00661                 continue;
00662         }
00663
00664         /* Set mass and volume mixing ratio... */
00665         if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
00666             atm->q[ctl->qnt_m][ip] =
00667                 ctl->bound_mass + ctl->bound_mass_trend * atm->time[ip];
00668         if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
00669             atm->q[ctl->qnt_vmr][ip] =
00670                 ctl->bound_vmr + ctl->bound_vmr_trend * atm->time[ip];
00671     }
00672 }
00673
00674 /*****
00675
00676 void module_convection(
00677     ctl_t * ctl,
00678     met_t * met0,
00679     met_t * met1,
00680     atm_t * atm,
00681     double *dt,
00682     double *rs) {
00683
00684     /* Set timer... */
00685     SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);
00686
00687     /* Create random numbers... */
00688     module_rng(rs, (size_t) atm->np, 0);
00689
00690     const int np = atm->np;
00691 #ifdef _OPENACC
00692 #pragma acc data present(ctl, met0, met1, atm, dt, rs)
00693 #pragma acc parallel loop independent gang vector
00694 #else
00695 #pragma omp parallel for default(shared)
00696 #endif
00697     for (int ip = 0; ip < np; ip++)
00698         if (dt[ip] != 0) {
00699
00700             double cape, cin, pel, ps;
00701
00702             /* Interpolate CAPE... */
00703             INTPOL_INIT;
00704             INTPOL_2D(cape, 1);
00705
00706             /* Check threshold... */
00707             if (isfinite(cape) && cape >= ctl->conv_cape) {
00708
00709                 /* Check CIN... */
00710                 if (ctl->conv_cin > 0) {
00711                     INTPOL_2D(cin, 0);
00712                     if (isfinite(cin) && cin >= ctl->conv_cin)
00713                         continue;
00714                 }
00715
00716                 /* Interpolate equilibrium level... */
00717                 INTPOL_2D(pel, 0);
00718
00719                 /* Check whether particle is above cloud top... */
00720                 if (!isfinite(pel) || atm->p[ip] < pel)
00721                     continue;
00722
00723                 /* Set pressure range for mixing... */
00724                 double pbot = atm->p[ip];
00725                 double ptop = atm->p[ip];
00726                 if (ctl->conv_mix_bot == 1) {
00727                     INTPOL_2D(ps, 0);
00728                     pbot = ps;
00729                 }

```

```

00730     if (ctl->conv_mix_top == 1)
00731         ptop = pel;
00732
00733     /* Limit vertical velocity... */
00734     if (ctl->conv_wmax > 0 || ctl->conv_wcape) {
00735         double z = Z(atm->p[ip]);
00736         double wmax = (ctl->conv_wcape) ? sqrt(2. * cape) : ctl->conv_wmax;
00737         double pmax = P(z - wmax * dt[ip] / 1000.);
00738         double pmin = P(z + wmax * dt[ip] / 1000.);
00739         ptop = GSL_MAX(ptop, pmin);
00740         pbot = GSL_MIN(pbot, pmax);
00741     }
00742
00743     /* Vertical mixing based on pressure... */
00744     if (ctl->conv_mix == 0)
00745         atm->p[ip] = pbot + (ptop - pbot) * rs[ip];
00746
00747     /* Vertical mixing based on density... */
00748     else if (ctl->conv_mix == 1) {
00749
00750         /* Get density range... */
00751         double tbot, ttop;
00752         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00753             pbot, atm->lon[ip], atm->lat[ip], &tbot,
00754             ci, cw, 1);
00755         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00756             ptop, atm->lon[ip], atm->lat[ip], &ttop,
00757             ci, cw, 1);
00758         double rhobot = pbot / tbot;
00759         double rhotop = ptop / ttop;
00760
00761         /* Get new density... */
00762         double lrho = log(rhobot + (rhotop - rhobot) * rs[ip]);
00763
00764         /* Find pressure... */
00765         double lrhobot = log(rhobot);
00766         double lrhotop = log(rhotop);
00767         double lpbot = log(pbot);
00768         double lptop = log(ptop);
00769         atm->p[ip] = exp(LIN(lrhobot, lpbot, lrhotop, lptop, lrho));
00770     }
00771 }
00772 }
00773 }
00774
00775 /*****
00776
00777 void module_decay(
00778     ctl_t * ctl,
00779     clim_t * clim,
00780     atm_t * atm,
00781     double *dt) {
00782
00783     /* Set timer... */
00784     SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);
00785
00786     /* Check quantity flags... */
00787     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00788         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00789
00790     const int np = atm->np;
00791 #ifdef _OPENACC
00792 #pragma acc data present(ctl, clim, atm, dt)
00793 #pragma acc parallel loop independent gang vector
00794 #else
00795 #pragma omp parallel for default(shared)
00796 #endif
00797     for (int ip = 0; ip < np; ip++)
00798         if (dt[ip] != 0) {
00799
00800             /* Get weighting factor... */
00801             double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00802
00803             /* Set lifetime... */
00804             double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00805
00806             /* Calculate exponential decay... */
00807             double aux = exp(-dt[ip] / tdec);
00808             if (ctl->qnt_m >= 0) {
00809                 if (ctl->qnt_mloss_decay >= 0)
00810                     atm->q[ctl->qnt_mloss_decay][ip]
00811                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
00812                 atm->q[ctl->qnt_m][ip] *= aux;
00813             }
00814             if (ctl->qnt_vmr >= 0)
00815                 atm->q[ctl->qnt_vmr][ip] *= aux;
00816         }

```



```

00817 }
00818
00819 /*****
00820
00821 void module_diffusion_meso(
00822     ctl_t * ctl,
00823     met_t * met0,
00824     met_t * met1,
00825     atm_t * atm,
00826     cache_t * cache,
00827     double *dt,
00828     double *rs) {
00829
00830     /* Set timer... */
00831     SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);
00832
00833     /* Create random numbers... */
00834     module_rng(rs, 3 * (size_t) atm->np, 1);
00835
00836     const int np = atm->np;
00837     #ifdef _OPENACC
00838     #pragma acc data present(ctl, met0, met1, atm, cache, dt, rs)
00839     #pragma acc parallel loop independent gang vector
00840     #else
00841     #pragma omp parallel for default(shared)
00842     #endif
00843     for (int ip = 0; ip < np; ip++)
00844         if (dt[ip] != 0) {
00845
00846             /* Get indices... */
00847             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00848             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00849             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00850
00851             /* Get standard deviations of local wind data... */
00852             float umean = 0, usig = 0, vmean = 0, vsig = 0, wmean = 0, wsig = 0;
00853             for (int i = 0; i < 2; i++)
00854                 for (int j = 0; j < 2; j++)
00855                     for (int k = 0; k < 2; k++) {
00856             #ifdef UVW
00857                 umean += met0->uvw[ix + i][iy + j][iz + k][0];
00858                 usig += SQR(met0->uvw[ix + i][iy + j][iz + k][0]);
00859                 vmean += met0->uvw[ix + i][iy + j][iz + k][1];
00860                 vsig += SQR(met0->uvw[ix + i][iy + j][iz + k][1]);
00861                 wmean += met0->uvw[ix + i][iy + j][iz + k][2];
00862                 wsig += SQR(met0->uvw[ix + i][iy + j][iz + k][2]);
00863
00864                 umean += met1->uvw[ix + i][iy + j][iz + k][0];
00865                 usig += SQR(met1->uvw[ix + i][iy + j][iz + k][0]);
00866                 vmean += met1->uvw[ix + i][iy + j][iz + k][1];
00867                 vsig += SQR(met1->uvw[ix + i][iy + j][iz + k][1]);
00868                 wmean += met1->uvw[ix + i][iy + j][iz + k][2];
00869                 wsig += SQR(met1->uvw[ix + i][iy + j][iz + k][2]);
00870             #else
00871                 umean += met0->u[ix + i][iy + j][iz + k];
00872                 usig += SQR(met0->u[ix + i][iy + j][iz + k]);
00873                 vmean += met0->v[ix + i][iy + j][iz + k];
00874                 vsig += SQR(met0->v[ix + i][iy + j][iz + k]);
00875                 wmean += met0->w[ix + i][iy + j][iz + k];
00876                 wsig += SQR(met0->w[ix + i][iy + j][iz + k]);
00877
00878                 umean += met1->u[ix + i][iy + j][iz + k];
00879                 usig += SQR(met1->u[ix + i][iy + j][iz + k]);
00880                 vmean += met1->v[ix + i][iy + j][iz + k];
00881                 vsig += SQR(met1->v[ix + i][iy + j][iz + k]);
00882                 wmean += met1->w[ix + i][iy + j][iz + k];
00883                 wsig += SQR(met1->w[ix + i][iy + j][iz + k]);
00884             #endif
00885             }
00886             usig = usig / 16.f - SQR(umean / 16.f);
00887             usig = (usig > 0 ? sqrtf(usig) : 0);
00888             vsig = vsig / 16.f - SQR(vmean / 16.f);
00889             vsig = (vsig > 0 ? sqrtf(vsig) : 0);
00890             wsig = wsig / 16.f - SQR(wmean / 16.f);
00891             wsig = (wsig > 0 ? sqrtf(wsig) : 0);
00892
00893             /* Set temporal correlations for mesoscale fluctuations... */
00894             double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00895             double r2 = sqrt(1 - r * r);
00896
00897             /* Calculate horizontal mesoscale wind fluctuations... */
00898             if (ctl->turb_mesox > 0) {
00899                 cache->uvw[ip][0] =
00900                     (float) (r * cache->uvw[ip][0] +
00901                             r2 * rs[3 * ip] * ctl->turb_mesox * usig);
00902                 atm->lon[ip] +=
00903                     DX2DEG(cache->uvw[ip][0] * dt[ip] / 1000., atm->lat[ip]);

```

```

00904
00905     cache->uwp[ip][1] =
00906         (float) (r * cache->uwp[ip][1] +
00907             r2 * rs[3 * ip + 1] * ctl->turb_mesox * vsig);
00908     atm->lat[ip] += DY2DEG(cache->uwp[ip][1] * dt[ip] / 1000.);
00909 }
00910
00911 /* Calculate vertical mesoscale wind fluctuations... */
00912 if (ctl->turb_mesoz > 0) {
00913     cache->uwp[ip][2] =
00914         (float) (r * cache->uwp[ip][2] +
00915             r2 * rs[3 * ip + 2] * ctl->turb_mesoz * wsig);
00916     atm->p[ip] += cache->uwp[ip][2] * dt[ip];
00917 }
00918 }
00919 }
00920
00921 /*****
00922
00923 void module_diffusion_turb(
00924     ctl_t * ctl,
00925     clim_t * clim,
00926     atm_t * atm,
00927     double *dt,
00928     double *rs) {
00929
00930     /* Set timer... */
00931     SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);
00932
00933     /* Create random numbers... */
00934     module_rng(rs, 3 * (size_t) atm->np, 1);
00935
00936     const int np = atm->np;
00937 #ifdef _OPENACC
00938 #pragma acc data present(ctl, clim, atm, dt, rs)
00939 #pragma acc parallel loop independent gang vector
00940 #else
00941 #pragma omp parallel for default(shared)
00942 #endif
00943     for (int ip = 0; ip < np; ip++)
00944         if (dt[ip] != 0) {
00945
00946             /* Get weighting factor... */
00947             double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00948
00949             /* Set diffusivity... */
00950             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00951             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00952
00953             /* Horizontal turbulent diffusion... */
00954             if (dx > 0) {
00955                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00956                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00957                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00958             }
00959
00960             /* Vertical turbulent diffusion... */
00961             if (dz > 0) {
00962                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00963                 atm->p[ip] += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00964             }
00965         }
00966 }
00967
00968 /*****
00969
00970 void module_dry_deposition(
00971     ctl_t * ctl,
00972     met_t * met0,
00973     met_t * met1,
00974     atm_t * atm,
00975     double *dt) {
00976
00977     /* Set timer... */
00978     SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);
00979
00980     /* Depth of the surface layer [hPa]. */
00981     const double dp = 30.;
00982
00983     /* Check quantity flags... */
00984     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00985         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00986
00987     const int np = atm->np;
00988 #ifdef _OPENACC
00989 #pragma acc data present(ctl, met0, met1, atm, dt)
00990 #pragma acc parallel loop independent gang vector

```

```

00991 #else
00992 #pragma omp parallel for default(shared)
00993 #endif
00994     for (int ip = 0; ip < np; ip++)
00995         if (dt[ip] != 0) {
00996             double ps, t, v_dep;
00997
00998             /* Get surface pressure... */
00999             INTPOL_INIT;
01000             INTPOL_2D(ps, 1);
01001
01002             /* Check whether particle is above the surface layer... */
01003             if (atm->p[ip] < ps - dp)
01004                 continue;
01005
01006             /* Set depth of surface layer... */
01007             double dz = 1000. * (Z(ps - dp) - Z(ps));
01008
01009             /* Calculate sedimentation velocity for particles... */
01010             if (ctl->qnt_rp > 0 && ctl->qnt_rhop > 0) {
01011                 /* Get temperature... */
01012                 INTPOL_3D(t, 1);
01013
01014                 /* Set deposition velocity... */
01015                 v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01016                             atm->q[ctl->qnt_rhop][ip]);
01017             }
01018
01019             /* Use explicit sedimentation velocity for gases... */
01020             else
01021                 v_dep = ctl->dry_depo[0];
01022
01023             /* Calculate loss of mass based on deposition velocity... */
01024             double aux = exp(-dt[ip] * v_dep / dz);
01025             if (ctl->qnt_m >= 0) {
01026                 if (ctl->qnt_mloss_dry >= 0)
01027                     atm->q[ctl->qnt_mloss_dry][ip]
01028                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01029                 atm->q[ctl->qnt_m][ip] *= aux;
01030             }
01031             if (ctl->qnt_vmr >= 0)
01032                 atm->q[ctl->qnt_vmr][ip] *= aux;
01033         }
01034     }
01035 }
01036
01037 /*****
01038
01039 void module_isosurf_init(
01040     ctl_t * ctl,
01041     met_t * met0,
01042     met_t * met1,
01043     atm_t * atm,
01044     cache_t * cache) {
01045     FILE *in;
01046
01047     char line[LEN];
01048
01049     double t;
01050
01051     /* Set timer... */
01052     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01053
01054     /* Init... */
01055     INTPOL_INIT;
01056
01057     /* Save pressure... */
01058     if (ctl->isosurf == 1)
01059         for (int ip = 0; ip < atm->np; ip++)
01060             cache->iso_var[ip] = atm->p[ip];
01061
01062     /* Save density... */
01063     else if (ctl->isosurf == 2)
01064         for (int ip = 0; ip < atm->np; ip++) {
01065             INTPOL_3D(t, 1);
01066             cache->iso_var[ip] = atm->p[ip] / t;
01067         }
01068
01069     /* Save potential temperature... */
01070     else if (ctl->isosurf == 3)
01071         for (int ip = 0; ip < atm->np; ip++) {
01072             INTPOL_3D(t, 1);
01073             cache->iso_var[ip] = THETA(atm->p[ip], t);
01074         }
01075     }
01076 }
01077

```

```

01078  /* Read balloon pressure data... */
01079  else if (ctl->isosurf == 4) {
01080
01081      /* Write info... */
01082      LOG(1, "Read balloon pressure data: %s", ctl->balloon);
01083
01084      /* Open file... */
01085      if (!(in = fopen(ctl->balloon, "r")))
01086          ERRMSG("Cannot open file!");
01087
01088      /* Read pressure time series... */
01089      while (fgets(line, LEN, in))
01090          if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
01091                    &(cache->iso_ps[cache->iso_n])) == 2)
01092              if ((++cache->iso_n) > NP)
01093                  ERRMSG("Too many data points!");
01094
01095      /* Check number of points... */
01096      if (cache->iso_n < 1)
01097          ERRMSG("Could not read any data!");
01098
01099      /* Close file... */
01100      fclose(in);
01101  }
01102 }
01103
01104 /*****
01105
01106 void module_isosurf(
01107     ctl_t * ctl,
01108     met_t * met0,
01109     met_t * met1,
01110     atm_t * atm,
01111     cache_t * cache) {
01112
01113     /* Set timer... */
01114     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01115
01116     const int np = atm->np;
01117     #ifdef _OPENACC
01118     #pragma acc data present(ctl, met0, met1, atm, cache)
01119     #pragma acc parallel loop independent gang vector
01120     #else
01121     #pragma omp parallel for default(shared)
01122     #endif
01123     for (int ip = 0; ip < np; ip++) {
01124
01125         double t;
01126
01127         /* Init... */
01128         INTPOL_INIT;
01129
01130         /* Restore pressure... */
01131         if (ctl->isosurf == 1)
01132             atm->p[ip] = cache->iso_var[ip];
01133
01134         /* Restore density... */
01135         else if (ctl->isosurf == 2) {
01136             INTPOL_3D(t, 1);
01137             atm->p[ip] = cache->iso_var[ip] * t;
01138         }
01139
01140         /* Restore potential temperature... */
01141         else if (ctl->isosurf == 3) {
01142             INTPOL_3D(t, 1);
01143             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
01144         }
01145
01146         /* Interpolate pressure... */
01147         else if (ctl->isosurf == 4) {
01148             if (atm->time[ip] <= cache->iso_ts[0])
01149                 atm->p[ip] = cache->iso_ps[0];
01150             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
01151                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
01152             else {
01153                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
01154                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
01155                                cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
01156                                atm->time[ip]);
01157             }
01158         }
01159     }
01160 }
01161
01162 /*****
01163
01164 void module_meteo(

```

```

01165     ctl_t * ctl,
01166     clim_t * clim,
01167     met_t * met0,
01168     met_t * met1,
01169     atm_t * atm) {
01170
01171     /* Set timer... */
01172     SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);
01173
01174     /* Check quantity flags... */
01175     if (ctl->qnt_tsts >= 0)
01176         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01177             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01178
01179     const int np = atm->np;
01180 #ifdef _OPENACC
01181 #pragma acc data present(ctl, clim, met0, met1, atm)
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 < np; ip++) {
01187
01188         double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
01189             cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
01190
01191         /* Interpolate meteo data... */
01192         INTPOL_INIT;
01193         INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
01194
01195         /* Set quantities... */
01196         SET_ATM(qnt_ps, ps);
01197         SET_ATM(qnt_ts, ts);
01198         SET_ATM(qnt_zs, zs);
01199         SET_ATM(qnt_us, us);
01200         SET_ATM(qnt_vs, vs);
01201         SET_ATM(qnt_pbl, pbl);
01202         SET_ATM(qnt_pt, pt);
01203         SET_ATM(qnt_tt, tt);
01204         SET_ATM(qnt_zt, zt);
01205         SET_ATM(qnt_h2ot, h2ot);
01206         SET_ATM(qnt_z, z);
01207         SET_ATM(qnt_p, atm->p[ip]);
01208         SET_ATM(qnt_t, t);
01209         SET_ATM(qnt_rho, RHO(atm->p[ip], t));
01210         SET_ATM(qnt_u, u);
01211         SET_ATM(qnt_v, v);
01212         SET_ATM(qnt_w, w);
01213         SET_ATM(qnt_h2o, h2o);
01214         SET_ATM(qnt_o3, o3);
01215         SET_ATM(qnt_lwc, lwc);
01216         SET_ATM(qnt_iwc, iwc);
01217         SET_ATM(qnt_pct, pct);
01218         SET_ATM(qnt_pcb, pcb);
01219         SET_ATM(qnt_cl, cl);
01220         SET_ATM(qnt_plcl, plcl);
01221         SET_ATM(qnt_plfc, plfc);
01222         SET_ATM(qnt_pel, pel);
01223         SET_ATM(qnt_cape, cape);
01224         SET_ATM(qnt_cin, cin);
01225         SET_ATM(qnt_hno3,
01226             clim_hno3(clim, atm->time[ip], atm->lat[ip], atm->p[ip]));
01227         SET_ATM(qnt_oh,
01228             clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01229                 atm->lon[ip], atm->lat[ip]));
01230         SET_ATM(qnt_vh, sqrt(u * u + v * v));
01231         SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01232         SET_ATM(qnt_psat, PSAT(t));
01233         SET_ATM(qnt_psice, PSICE(t));
01234         SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
01235         SET_ATM(qnt_sh, SH(h2o));
01236         SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
01237         SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01238         SET_ATM(qnt_theta, THETA(atm->p[ip], t));
01239         SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
01240         SET_ATM(qnt_tvirt, TVIRT(t, h2o));
01241         SET_ATM(qnt_lapse, lapse_rate(t, h2o));
01242         SET_ATM(qnt_pv, pv);
01243         SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
01244         SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
01245         SET_ATM(qnt_tnat,
01246             nat_temperature(atm->p[ip], h2o,
01247                 clim_hno3(clim, atm->time[ip], atm->lat[ip],
01248                     atm->p[ip])));
01249         SET_ATM(qnt_tsts,
01250             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01251     }

```

```

01252 }
01253
01254 /*****
01255
01256 void module_oh_chem(
01257     ctl_t * ctl,
01258     clim_t * clim,
01259     met_t * met0,
01260     met_t * met1,
01261     atm_t * atm,
01262     double *dt) {
01263
01264     /* Set timer... */
01265     SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);
01266
01267     /* Check quantity flags... */
01268     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01269         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01270
01271     const int np = atm->np;
01272     #ifdef _OPENACC
01273     #pragma acc data present(ctl, clim, met0, met1, atm, dt)
01274     #pragma acc parallel loop independent gang vector
01275     #else
01276     #pragma omp parallel for default(shared)
01277     #endif
01278     for (int ip = 0; ip < np; ip++)
01279         if (dt[ip] != 0) {
01280
01281             /* Get temperature... */
01282             double t;
01283             INTPOL_INIT;
01284             INTPOL_3D(t, 1);
01285
01286             /* Use constant reaction rate... */
01287             double k = GSL_NAN;
01288             if (ctl->oh_chem_reaction == 1)
01289                 k = ctl->oh_chem[0];
01290
01291             /* Calculate bimolecular reaction rate... */
01292             else if (ctl->oh_chem_reaction == 2)
01293                 k = ctl->oh_chem[0] * exp(-(ctl->oh_chem[1] / t);
01294
01295             /* Calculate termolecular reaction rate... */
01296             if (ctl->oh_chem_reaction == 3) {
01297
01298                 /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
01299                 double M = 7.243e21 * (atm->p[ip] / 1000.) / t;
01300
01301                 /* Calculate rate coefficient for X + OH + M -> XOH + M
01302                    (JPL Publication 19-05) ... */
01303                 double k0 = ctl->oh_chem[0] *
01304                     (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
01305                 double ki = ctl->oh_chem[2] *
01306                     (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
01307                 double c = log10(k0 * M / ki);
01308                 k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01309             }
01310
01311             /* Calculate exponential decay... */
01312             double rate_coef =
01313                 k * clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01314                                     atm->lon[ip],
01315                                     atm->lat[ip]);
01316             double aux = exp(-dt[ip] * rate_coef);
01317             if (ctl->qnt_m >= 0) {
01318                 if (ctl->qnt_mloss_oh >= 0)
01319                     atm->q[ctl->qnt_mloss_oh][ip]
01320                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01321                 atm->q[ctl->qnt_m][ip] *= aux;
01322             }
01323             if (ctl->qnt_vmr >= 0)
01324                 atm->q[ctl->qnt_vmr][ip] *= aux;
01325         }
01326 }
01327
01328 /*****
01329
01330 void module_h2o2_chem(
01331     ctl_t * ctl,
01332     clim_t * clim,
01333     met_t * met0,
01334     met_t * met1,
01335     atm_t * atm,
01336     double *dt,
01337     double *rs) {
01338

```

```

01339  /* Set timer... */
01340  SELECT_TIMER("MODULE_H2O2CHEM", "PHYSICS", NVTX_GPU);
01341
01342  /* Check quantity flags... */
01343  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01344      ERRMSG("Module needs quantity mass or volume mixing ratio!");
01345  if (ctl->qnt_vmrimpl < 0)
01346      ERRMSG("Module needs quantity implicit volume mixing ratio!");
01347
01348  /* Create random numbers... */
01349  module_rng(rs, (size_t) atm->np, 0);
01350
01351  const int np = atm->np;
01352  #ifdef _OPENACC
01353  #pragma acc data present(clim,ctl,met0,met1,atm,dt,rs)
01354  #pragma acc parallel loop independent gang vector
01355  #else
01356  #pragma omp parallel for default(shared)
01357  #endif
01358  for (int ip = 0; ip < np; ip++)
01359      if (dt[ip] != 0) {
01360
01361          /* Check whether particle is inside cloud... */
01362          double lwc, iwc;
01363          INTPOL_INIT;
01364          INTPOL_3D(lwc, 1);
01365          INTPOL_3D(iwc, 0);
01366          if (!(lwc > 0 || iwc > 0))
01367              continue;
01368
01369          /* Check cloud cover... */
01370          if (rs[ip] > ctl->h2o2_chem_cc)
01371              continue;
01372
01373          /* Check implicit volume mixing ratio... */
01374          if (atm->q[ctl->qnt_vmrimpl][ip] == 0)
01375              continue;
01376
01377          /* Get temperature... */
01378          double t;
01379          INTPOL_3D(t, 0);
01380
01381          /* Reaction rate (Berglen et al., 2004)... */
01382          double k = 9.1e7 * exp(-29700 / RI * (1. / t - 1. / 298.15)); // Maass 1999 unit: M^(-2)
01383
01384          /* Henry constant of SO2... */
01385          double H_SO2 = 1.3e-2 * exp(2900 * (1. / t - 1. / 298.15)) * RI * t;
01386          double K_1S = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15)); // unit: M
01387
01388          /* Henry constant of H2O2... */
01389          double H_h2o2 = 8.3e2 * exp(7600 * (1 / t - 1 / 298.15)) * RI * t;
01390
01391          /* Concentration of H2O2 (Barth et al., 1989)... */
01392          double SO2 = atm->q[ctl->qnt_vmrimpl][ip] * 1e9; // vmr unit: ppbv
01393          double h2o2 = H_h2o2
01394              * clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip])
01395              * 0.59 * exp(-0.687 * SO2) * 1000 / 6.02214e23; // unit: M
01396
01397          /* Volume water content in cloud [m^3 m^(-3)]... */
01398          double rho_air = 100 * atm->p[ip] / (RA * t);
01399          double CWC = lwc * rho_air / 1000 + iwc * rho_air / 920;
01400
01401          /* Calculate exponential decay (Rolph et al., 1992)... */
01402          double rate_coef = k * K_1S * h2o2 * H_SO2 * CWC;
01403          double aux = exp(-dt[ip] * rate_coef);
01404          if (ctl->qnt_m >= 0) {
01405              if (ctl->qnt_mloss_h2o2 >= 0)
01406                  atm->q[ctl->qnt_mloss_h2o2][ip] +=
01407                      atm->q[ctl->qnt_m][ip] * (1 - aux);
01408                  atm->q[ctl->qnt_m][ip] *= aux;
01409              }
01410              if (ctl->qnt_vmr >= 0)
01411                  atm->q[ctl->qnt_vmr][ip] *= aux;
01412          }
01413      }
01414
01415  /*****
01416
01417  void module_position(
01418      ctl_t * ctl,
01419      met_t * met0,
01420      met_t * met1,
01421      atm_t * atm,
01422      double *dt) {
01423
01424      /* Set timer... */
01425      SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);

```

```

01426
01427     const int np = atm->np;
01428 #ifdef _OPENACC
01429 #pragma acc data present(met0, met1, atm, dt)
01430 #pragma acc parallel loop independent gang vector
01431 #else
01432 #pragma omp parallel for default(shared)
01433 #endif
01434     for (int ip = 0; ip < np; ip++)
01435         if (dt[ip] != 0) {
01436
01437             /* Init... */
01438             double ps;
01439             INTPOL_INIT;
01440
01441             /* Calculate modulo... */
01442             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01443             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01444
01445             /* Check latitude... */
01446             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01447                 if (atm->lat[ip] > 90) {
01448                     atm->lat[ip] = 180 - atm->lat[ip];
01449                     atm->lon[ip] += 180;
01450                 }
01451                 if (atm->lat[ip] < -90) {
01452                     atm->lat[ip] = -180 - atm->lat[ip];
01453                     atm->lon[ip] += 180;
01454                 }
01455             }
01456
01457             /* Check longitude... */
01458             while (atm->lon[ip] < -180)
01459                 atm->lon[ip] += 360;
01460             while (atm->lon[ip] >= 180)
01461                 atm->lon[ip] -= 360;
01462
01463             /* Check pressure... */
01464             if (atm->p[ip] < met0->p[met0->np - 1]) {
01465                 if (ctl->reflect)
01466                     atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
01467                 else
01468                     atm->p[ip] = met0->p[met0->np - 1];
01469             } else if (atm->p[ip] > 300.) {
01470                 INTPOL_2D(ps, 1);
01471                 if (atm->p[ip] > ps) {
01472                     if (ctl->reflect)
01473                         atm->p[ip] = 2. * ps - atm->p[ip];
01474                     else
01475                         atm->p[ip] = ps;
01476                 }
01477             }
01478         }
01479 }
01480
01481 /*****
01482
01483 void module_rng_init(
01484     int ntask) {
01485
01486     /* Initialize random number generator... */
01487 #ifdef _OPENACC
01488
01489     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT) !=
01490         CURAND_STATUS_SUCCESS)
01491         ERRMSG("Cannot create random number generator!");
01492     if (curandSetPseudoRandomGeneratorSeed(rng, ntask) != CURAND_STATUS_SUCCESS)
01493         ERRMSG("Cannot set seed for random number generator!");
01494     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01495         != CURAND_STATUS_SUCCESS)
01496         ERRMSG("Cannot set stream for random number generator!");
01497 #else
01498     else
01499
01500     gsl_rng_env_setup();
01501     if (omp_get_max_threads() > NTHREADS)
01502         ERRMSG("Too many threads!");
01503     for (int i = 0; i < NTHREADS; i++) {
01504         rng[i] = gsl_rng_alloc(gsl_rng_default);
01505         gsl_rng_set(rng[i],
01506             gsl_rng_default_seed + (long unsigned) (ntask * NTHREADS +
01507                 i));
01508     }
01509 #endif
01510 }
01511
01512

```



```

01513 /*****
01514
01515 void module_rng(
01516     double *rs,
01517     size_t n,
01518     int method) {
01519
01520 #ifdef _OPENACC
01521
01522 #pragma acc host_data use_device(rs)
01523 {
01524     /* Uniform distribution... */
01525     if (method == 0) {
01526         if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n)) !=
01527             CURAND_STATUS_SUCCESS)
01528             ERRMSG("Cannot create random numbers!");
01529     }
01530
01531     /* Normal distribution... */
01532     else if (method == 1) {
01533         if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0) !=
01534             CURAND_STATUS_SUCCESS)
01535             ERRMSG("Cannot create random numbers!");
01536     }
01537 }
01538 #else
01539
01540     /* Uniform distribution... */
01541     if (method == 0) {
01542 #pragma omp parallel for default(shared)
01543         for (size_t i = 0; i < n; ++i)
01544             rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01545     }
01546
01547     /* Normal distribution... */
01548     else if (method == 1) {
01549 #pragma omp parallel for default(shared)
01550         for (size_t i = 0; i < n; ++i)
01551             rs[i] = gsl_rng_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01552     }
01553 #endif
01554 }
01555
01556 /****
01557
01558 void module_sedi(
01559     ctl_t *ctl,
01560     met_t *met0,
01561     met_t *met1,
01562     atm_t *atm,
01563     double *dt) {
01564
01565     /* Set timer... */
01566     SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);
01567
01568     const int np = atm->np;
01569 #ifdef _OPENACC
01570 #pragma acc data present(ctl, met0, met1, atm, dt)
01571 #pragma acc parallel loop independent gang vector
01572 #else
01573 #pragma omp parallel for default(shared)
01574 #endif
01575     for (int ip = 0; ip < np; ip++)
01576         if (dt[ip] != 0) {
01577
01578             /* Get temperature... */
01579             double t;
01580             INTPOL_INIT;
01581             INTPOL_3D(t, 1);
01582
01583             /* Sedimentation velocity... */
01584             double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01585                             atm->q[ctl->qnt_rhop][ip]);
01586
01587             /* Calculate pressure change... */
01588             atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01589         }
01590 }
01591
01592 /****
01593
01594 void module_sort(
01595     ctl_t *ctl,
01596     met_t *met0,
01597     atm_t *atm) {
01598
01599

```

```

01600  /* Set timer... */
01601  SELECT_TIMER("MODULE_SORT", "PHYSICS", NVTX_GPU);
01602
01603  /* Allocate... */
01604  const int np = atm->np;
01605  double *restrict const a = (double *) malloc((size_t) np * sizeof(double));
01606  int *restrict const p = (int *) malloc((size_t) np * sizeof(int));
01607
01608  #ifdef _OPENACC
01609  #pragma acc enter data create(a[0:np],p[0:np])
01610  #pragma acc data present(ctl,met0,atm,a,p)
01611  #endif
01612
01613  /* Get box index... */
01614  #ifdef _OPENACC
01615  #pragma acc parallel loop independent gang vector
01616  #else
01617  #pragma omp parallel for default(shared)
01618  #endif
01619  for (int ip = 0; ip < np; ip++) {
01620      a[ip] =
01621          (double) ((locate_reg(met0->lon, met0->nx, atm->lon[ip]) * met0->ny +
01622                  locate_reg(met0->lat, met0->ny,
01623                          atm->lat[ip])) * met0->np + locate_irr(met0->p,
01624                              met0->np,
01625                              atm->p
01626                              [ip]));
01627      p[ip] = ip;
01628  }
01629
01630  /* Sorting... */
01631  #ifdef _OPENACC
01632  {
01633  #ifdef THRUST
01634      {
01635      #pragma acc host_data use_device(a, p)
01636      thrustSortWrapper(a, np, p);
01637      }
01638  #else
01639      {
01640      #pragma acc update host(a[0:np], p[0:np])
01641      #pragma omp parallel
01642      {
01643      #pragma omp single nowait
01644      quicksort(a, p, 0, np - 1);
01645      }
01646      #pragma acc update device(a[0:np], p[0:np])
01647      }
01648      #endif
01649      }
01650  #else
01651  {
01652  #pragma omp parallel
01653  {
01654  #pragma omp single nowait
01655  quicksort(a, p, 0, np - 1);
01656  }
01657  }
01658  #endif
01659
01660  /* Sort data... */
01661  module_sort_help(atm->time, p, np);
01662  module_sort_help(atm->p, p, np);
01663  module_sort_help(atm->lon, p, np);
01664  module_sort_help(atm->lat, p, np);
01665  for (int iq = 0; iq < ctl->nq; iq++)
01666      module_sort_help(atm->q[iq], p, np);
01667
01668  /* Free... */
01669  #ifdef _OPENACC
01670  #pragma acc exit data delete(a,p)
01671  #endif
01672  free(a);
01673  free(p);
01674  }
01675
01676  /*****
01677
01678 void module_sort_help(
01679     double *a,
01680     int *p,
01681     int np) {
01682
01683     /* Allocate... */
01684     double *restrict const help =
01685         (double *) malloc((size_t) np * sizeof(double));
01686

```

```

01687  /* Reordering of array... */
01688  #ifdef _OPENACC
01689  #pragma acc enter data create(help[0:np])
01690  #pragma acc data present(a,p,help)
01691  #pragma acc parallel loop independent gang vector
01692  #endif
01693  for (int ip = 0; ip < np; ip++)
01694      help[ip] = a[p[ip]];
01695  #ifdef _OPENACC
01696  #pragma acc parallel loop independent gang vector
01697  #endif
01698  for (int ip = 0; ip < np; ip++)
01699      a[ip] = help[ip];
01700
01701  /* Free... */
01702  #ifdef _OPENACC
01703  #pragma acc exit data delete(help)
01704  #endif
01705  free(help);
01706  }
01707
01708  /*****
01709
01710 void module_timesteps(
01711     ctl_t * ctl,
01712     atm_t * atm,
01713     double *dt,
01714     double t) {
01715
01716     /* Set timer... */
01717     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01718
01719     const int np = atm->np;
01720     #ifdef _OPENACC
01721     #pragma acc data present(ctl, atm, dt)
01722     #pragma acc parallel loop independent gang vector
01723     #else
01724     #pragma omp parallel for default(shared)
01725     #endif
01726     for (int ip = 0; ip < np; ip++) {
01727         if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
01728             && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
01729             && ctl->direction * (atm->time[ip] - t) < 0))
01730             dt[ip] = t - atm->time[ip];
01731         else
01732             dt[ip] = 0.0;
01733     }
01734 }
01735
01736 /*****
01737
01738 void module_timesteps_init(
01739     ctl_t * ctl,
01740     atm_t * atm) {
01741
01742     /* Set timer... */
01743     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01744
01745     /* Set start time... */
01746     if (ctl->direction == 1) {
01747         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01748         if (ctl->t_stop > 1e99)
01749             ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01750     } else {
01751         ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01752         if (ctl->t_stop > 1e99)
01753             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01754     }
01755
01756     /* Check time interval... */
01757     if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
01758         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
01759
01760     /* Round start time... */
01761     if (ctl->direction == 1)
01762         ctl->t_start = floor(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01763     else
01764         ctl->t_start = ceil(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01765 }
01766
01767 /*****
01768
01769 void module_wet_deposition(
01770     ctl_t * ctl,
01771     met_t * met0,
01772     met_t * met1,
01773     atm_t * atm,

```

```

01774 double *dt) {
01775
01776 /* Set timer... */
01777 SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);
01778
01779 /* Check quantity flags... */
01780 if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01781     ERRMSG("Module needs quantity mass or volume mixing ratio!");
01782
01783 const int np = atm->np;
01784 #ifdef _OPENACC
01785 #pragma acc data present(ctl, met0, met1, atm, dt)
01786 #pragma acc parallel loop independent gang vector
01787 #else
01788 #pragma omp parallel for default(shared)
01789 #endif
01790 for (int ip = 0; ip < np; ip++)
01791     if (dt[ip] != 0) {
01792
01793         double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;
01794
01795         /* Check whether particle is below cloud top... */
01796         INTPOL_INIT;
01797         INTPOL_2D(pct, 1);
01798         if (!isfinite(pct) || atm->p[ip] <= pct)
01799             continue;
01800
01801         /* Get cloud bottom pressure... */
01802         INTPOL_2D(pcb, 0);
01803
01804         /* Estimate precipitation rate (Pisso et al., 2019)... */
01805         INTPOL_2D(cl, 0);
01806         double Is =
01807             pow(1. / ctl->wet_depo_pre[0] * cl, 1. / ctl->wet_depo_pre[1]);
01808         if (Is < 0.01)
01809             continue;
01810
01811         /* Check whether particle is inside or below cloud... */
01812         INTPOL_3D(lwc, 1);
01813         INTPOL_3D(iwc, 0);
01814         int inside = (iwc > 0 || lwc > 0);
01815
01816         /* Get temperature... */
01817         INTPOL_3D(t, 0);
01818
01819         /* Calculate in-cloud scavenging coefficient... */
01820         if (inside) {
01821
01822             /* Calculate retention factor... */
01823             double eta;
01824             if (t > 273.15)
01825                 eta = 1;
01826             else if (t <= 238.15)
01827                 eta = ctl->wet_depo_ic_ret_ratio;
01828             else
01829                 eta = LIN(273.15, 1, 238.15, ctl->wet_depo_ic_ret_ratio, t);
01830
01831             /* Use exponential dependency for particles ... */
01832             if (ctl->wet_depo_ic_a > 0)
01833                 lambda = ctl->wet_depo_ic_a * pow(Is, ctl->wet_depo_ic_b) * eta;
01834
01835             /* Use Henry's law for gases... */
01836             else if (ctl->wet_depo_ic_h[0] > 0) {
01837
01838                 /* Get Henry's constant (Sander, 2015)... */
01839                 h = ctl->wet_depo_ic_h[0]
01840                     * exp(ctl->wet_depo_ic_h[1] * (1. / t - 1. / 298.15));
01841
01842                 /* Use effective Henry's constant for SO2
01843                    (Berglen, 2004; Simpson, 2012)... */
01844                 if (ctl->wet_depo_ic_h[2] > 0) {
01845                     double H_ion = pow(10, ctl->wet_depo_ic_h[2] * (-1));
01846                     double K_1 = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));
01847                     double K_2 = 6e-8 * exp(1.12e3 * (1. / t - 1. / 298.15));
01848                     h *= (1 + K_1 / H_ion + K_1 * K_2 / pow(H_ion, 2));
01849                 }
01850
01851                 /* Estimate depth of cloud layer... */
01852                 dz = 1e3 * (Z(pct) - Z(pcb));
01853
01854                 /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01855                 lambda = h * RI * t * Is / 3.6e6 / dz * eta;
01856             }
01857         }
01858
01859         /* Calculate below-cloud scavenging coefficient... */
01860     else {

```

```

01861
01862     /* Calculate retention factor... */
01863     double eta;
01864     if (t > 270)
01865         eta = 1;
01866     else
01867         eta = ctl->wet_depo_bc_ret_ratio;
01868
01869     /* Use exponential dependency for particles... */
01870     if (ctl->wet_depo_bc_a > 0)
01871         lambda = ctl->wet_depo_bc_a * pow(Is, ctl->wet_depo_bc_b) * eta;
01872
01873     /* Use Henry's law for gases... */
01874     else if (ctl->wet_depo_bc_h[0] > 0) {
01875
01876         /* Get Henry's constant (Sander, 2015)... */
01877         h = ctl->wet_depo_bc_h[0]
01878             * exp(ctl->wet_depo_bc_h[1] * (1. / t - 1. / 298.15));
01879
01880         /* Estimate depth of cloud layer... */
01881         dz = 1e3 * (Z(pct) - Z(pcb));
01882
01883         /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01884         lambda = h * RI * t * Is / 3.6e6 / dz * eta;
01885     }
01886 }
01887
01888     /* Calculate exponential decay of mass... */
01889     double aux = exp(-dt[ip] * lambda);
01890     if (ctl->qnt_m >= 0) {
01891         if (ctl->qnt_mloss_wet >= 0)
01892             atm->q[ctl->qnt_mloss_wet][ip]
01893                 += atm->q[ctl->qnt_m][ip] * (1 - aux);
01894         atm->q[ctl->qnt_m][ip] *= aux;
01895     }
01896     if (ctl->qnt_vmr >= 0)
01897         atm->q[ctl->qnt_vmr][ip] *= aux;
01898 }
01899 }
01900
01901 /*****
01902
01903 void write_output(
01904     const char *dirname,
01905     ctl_t *ctl,
01906     met_t *met0,
01907     met_t *met1,
01908     atm_t *atm,
01909     double t) {
01910
01911     char ext[10], filename[2 * LEN];
01912
01913     double r;
01914
01915     int year, mon, day, hour, min, sec;
01916
01917     /* Get time... */
01918     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01919
01920     /* Update host... */
01921     #ifdef _OPENACC
01922     if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
01923         || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
01924         || (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0)
01925         || (ctl->csi_basename[0] != '-' || ctl->prof_basename[0] != '-')
01926         || (ctl->sample_basename[0] != '-' || ctl->stat_basename[0] != '-')) {
01927         SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
01928         #pragma acc update host(atm[:1])
01929     }
01930     #endif
01931
01932     /* Write atmospheric data... */
01933     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01934         if (ctl->atm_type == 0)
01935             sprintf(ext, "tab");
01936         else if (ctl->atm_type == 1)
01937             sprintf(ext, "bin");
01938         else if (ctl->atm_type == 2)
01939             sprintf(ext, "nc");
01940         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.%s",
01941             dirname, ctl->atm_basename, year, mon, day, hour, min, ext);
01942         write_atm(filename, ctl, atm, t);
01943     }
01944
01945     /* Write gridded data... */
01946     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01947         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.%s",

```

```

01948         dirname, ctl->grid_basename, year, mon, day, hour, min,
01949         ctl->grid_type == 0 ? "tab" : "nc");
01950     write_grid(filename, ctl, met0, met1, atm, t);
01951 }
01952
01953 /* Write CSI data... */
01954 if (ctl->csi_basename[0] != '-') {
01955     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01956     write_csi(filename, ctl, atm, t);
01957 }
01958
01959 /* Write ensemble data... */
01960 if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
01961     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01962             dirname, ctl->ens_basename, year, mon, day, hour, min);
01963     write_ens(filename, ctl, atm, t);
01964 }
01965
01966 /* Write profile data... */
01967 if (ctl->prof_basename[0] != '-') {
01968     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01969     write_prof(filename, ctl, met0, met1, atm, t);
01970 }
01971
01972 /* Write sample data... */
01973 if (ctl->sample_basename[0] != '-') {
01974     sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
01975     write_sample(filename, ctl, met0, met1, atm, t);
01976 }
01977
01978 /* Write station data... */
01979 if (ctl->stat_basename[0] != '-') {
01980     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01981     write_station(filename, ctl, atm, t);
01982 }
01983 }

```

## 5.47 tropo.c File Reference

Create tropopause data set from meteorological data.

```
#include "libtrac.h"
```

### Functions

- void [get\\_tropo](#) (int met\_tropo, [ctl\\_t](#) \*ctl, [clim\\_t](#) \*clim, [met\\_t](#) \*met, double \*lons, int nx, double \*lats, int ny, double \*pt, double \*zt, double \*tt, double \*qt)
- int [main](#) (int argc, char \*argv[])

#### 5.47.1 Detailed Description

Create tropopause data set from meteorological data.

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

#### 5.47.2 Function Documentation

**5.47.2.1 get\_tropo()** void get\_tropo (

```

    int met_tropo,
    ctl_t * ctl,
    clim_t * clim,
    met_t * met,
    double * lons,
    int nx,
    double * lats,
    int ny,
    double * pt,
    double * zt,
    double * tt,
    double * qt )

```

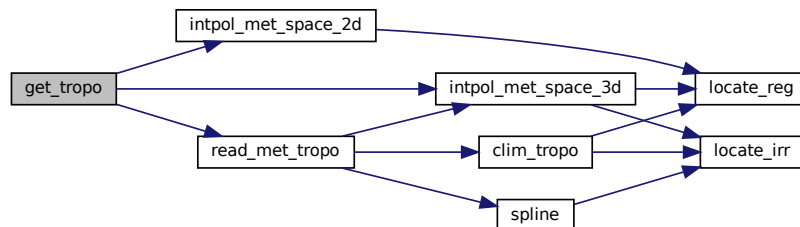
Definition at line 243 of file [tropo.c](#).

```

00255     {
00256
00257     INTPOL_INIT;
00258
00259     ctl->met_tropo = met_tropo;
00260     read_met_tropo(ctl, clim, met);
00261     #pragma omp parallel for default(shared) private(ci,cw)
00262     for (int ix = 0; ix < nx; ix++)
00263     for (int iy = 0; iy < ny; iy++) {
00264         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00265                             &pt[iy * nx + ix], ci, cw, 1);
00266         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00267                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00268         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00269                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00270         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00271                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00272     }
00273 }

```

Here is the call graph for this function:



**5.47.2.2 main()** int main (

```

    int argc,
    char * argv[] )

```

Definition at line 49 of file [tropo.c](#).

```

00051     {
00052
00053     ctl_t ctl;
00054
00055     clim_t *clim;
00056
00057     met_t *met;
00058

```

```

00059 static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00060 lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat;
00061
00062 static int init, i, nx, ny, nt, ncid, varid, dims[3], h2o;
00063
00064 static size_t count[10], start[10];
00065
00066 /* Allocate... */
00067 ALLOC(clim, clim_t, 1);
00068 ALLOC(met, met_t, 1);
00069
00070 /* Check arguments... */
00071 if (argc < 4)
00072     ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00073
00074 /* Read control parameters... */
00075 read_ctl(argv[1], argc, argv, &ctl);
00076 lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00077 lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00078 dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00079 lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00080 lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00081 dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00082 h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00083
00084 /* Read climatological data... */
00085 read_clim(&ctl, clim);
00086
00087 /* Loop over files... */
00088 for (i = 3; i < argc; i++) {
00089
00090     /* Read meteorological data... */
00091     ctl.met_tropo = 0;
00092     if (!read_met(argv[i], &ctl, clim, met))
00093         continue;
00094
00095     /* Set horizontal grid... */
00096     if (!init) {
00097         init = 1;
00098
00099         /* Get grid... */
00100         if (dlon <= 0)
00101             dlon = fabs(met->lon[1] - met->lon[0]);
00102         if (dlat <= 0)
00103             dlat = fabs(met->lat[1] - met->lat[0]);
00104         if (lon0 < -360 && lon1 > 360) {
00105             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00106             lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00107         }
00108         nx = ny = 0;
00109         for (lon = lon0; lon <= lon1; lon += dlon) {
00110             lons[nx] = lon;
00111             if ((++nx) > EX)
00112                 ERRMSG("Too many longitudes!");
00113         }
00114         if (lat0 < -90 && lat1 > 90) {
00115             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00116             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00117         }
00118         for (lat = lat0; lat <= lat1; lat += dlat) {
00119             lats[ny] = lat;
00120             if ((++ny) > EY)
00121                 ERRMSG("Too many latitudes!");
00122         }
00123
00124         /* Create netCDF file... */
00125         LOG(1, "Write tropopause data file: %s", argv[2]);
00126         NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00127
00128         /* Create dimensions... */
00129         NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00130         NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00131         NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00132
00133         /* Create variables... */
00134         NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time",
00135             "seconds since 2000-01-01 00:00:00 UTC");
00136         NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[1], "latitude", "degrees_north");
00137         NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[2], "longitude", "degrees_east");
00138
00139         NC_DEF_VAR("clp_z", NC_FLOAT, 3, &dims[0], "cold point height", "km");
00140         NC_DEF_VAR("clp_p", NC_FLOAT, 3, &dims[0], "cold point pressure",
00141             "hPa");
00142         NC_DEF_VAR("clp_t", NC_FLOAT, 3, &dims[0], "cold point temperature",
00143             "K");
00144         if (h2o)
00145             NC_DEF_VAR("clp_q", NC_FLOAT, 3, &dims[0], "cold point water vapor",

```



```

00146         "ppv");
00147
00148     NC_DEF_VAR("dyn_z", NC_FLOAT, 3, &dims[0],
00149 "dynamical tropopause height", "km");
00150     NC_DEF_VAR("dyn_p", NC_FLOAT, 3, &dims[0],
00151 "dynamical tropopause pressure", "hPa");
00152     NC_DEF_VAR("dyn_t", NC_FLOAT, 3, &dims[0],
00153 "dynamical tropopause temperature", "K");
00154     if (h2o)
00155         NC_DEF_VAR("dyn_q", NC_FLOAT, 3, &dims[0],
00156 "dynamical tropopause water vapor", "ppv");
00157
00158     NC_DEF_VAR("wmo_1st_z", NC_FLOAT, 3, &dims[0],
00159 "WMO 1st tropopause height", "km");
00160     NC_DEF_VAR("wmo_1st_p", NC_FLOAT, 3, &dims[0],
00161 "WMO 1st tropopause pressure", "hPa");
00162     NC_DEF_VAR("wmo_1st_t", NC_FLOAT, 3, &dims[0],
00163 "WMO 1st tropopause temperature", "K");
00164     if (h2o)
00165         NC_DEF_VAR("wmo_1st_q", NC_FLOAT, 3, &dims[0],
00166 "WMO 1st tropopause water vapor", "ppv");
00167
00168     NC_DEF_VAR("wmo_2nd_z", NC_FLOAT, 3, &dims[0],
00169 "WMO 2nd tropopause height", "km");
00170     NC_DEF_VAR("wmo_2nd_p", NC_FLOAT, 3, &dims[0],
00171 "WMO 2nd tropopause pressure", "hPa");
00172     NC_DEF_VAR("wmo_2nd_t", NC_FLOAT, 3, &dims[0],
00173 "WMO 2nd tropopause temperature", "K");
00174     if (h2o)
00175         NC_DEF_VAR("wmo_2nd_q", NC_FLOAT, 3, &dims[0],
00176 "WMO 2nd tropopause water vapor", "ppv");
00177
00178     /* End definition... */
00179     NC(nc_enddef(ncid));
00180
00181     /* Write longitude and latitude... */
00182     NC_PUT_DOUBLE("lat", lats, 0);
00183     NC_PUT_DOUBLE("lon", lons, 0);
00184 }
00185
00186 /* Write time... */
00187 start[0] = (size_t) nt;
00188 count[0] = 1;
00189 start[1] = 0;
00190 count[1] = (size_t) ny;
00191 start[2] = 0;
00192 count[2] = (size_t) nx;
00193 NC_PUT_DOUBLE("time", &met->time, 1);
00194
00195 /* Get cold point... */
00196 get_tropo(2, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00197 NC_PUT_DOUBLE("clp_z", zt, 1);
00198 NC_PUT_DOUBLE("clp_p", pt, 1);
00199 NC_PUT_DOUBLE("clp_t", tt, 1);
00200 if (h2o)
00201     NC_PUT_DOUBLE("clp_q", qt, 1);
00202
00203 /* Get dynamical tropopause... */
00204 get_tropo(5, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00205 NC_PUT_DOUBLE("dyn_z", zt, 1);
00206 NC_PUT_DOUBLE("dyn_p", pt, 1);
00207 NC_PUT_DOUBLE("dyn_t", tt, 1);
00208 if (h2o)
00209     NC_PUT_DOUBLE("dyn_q", qt, 1);
00210
00211 /* Get WMO 1st tropopause... */
00212 get_tropo(3, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00213 NC_PUT_DOUBLE("wmo_1st_z", zt, 1);
00214 NC_PUT_DOUBLE("wmo_1st_p", pt, 1);
00215 NC_PUT_DOUBLE("wmo_1st_t", tt, 1);
00216 if (h2o)
00217     NC_PUT_DOUBLE("wmo_1st_q", qt, 1);
00218
00219 /* Get WMO 2nd tropopause... */
00220 get_tropo(4, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00221 NC_PUT_DOUBLE("wmo_2nd_z", zt, 1);
00222 NC_PUT_DOUBLE("wmo_2nd_p", pt, 1);
00223 NC_PUT_DOUBLE("wmo_2nd_t", tt, 1);
00224 if (h2o)
00225     NC_PUT_DOUBLE("wmo_2nd_q", qt, 1);
00226
00227 /* Increment time step counter... */
00228 nt++;
00229 }
00230
00231 /* Close file... */
00232 NC(nc_close(ncid));

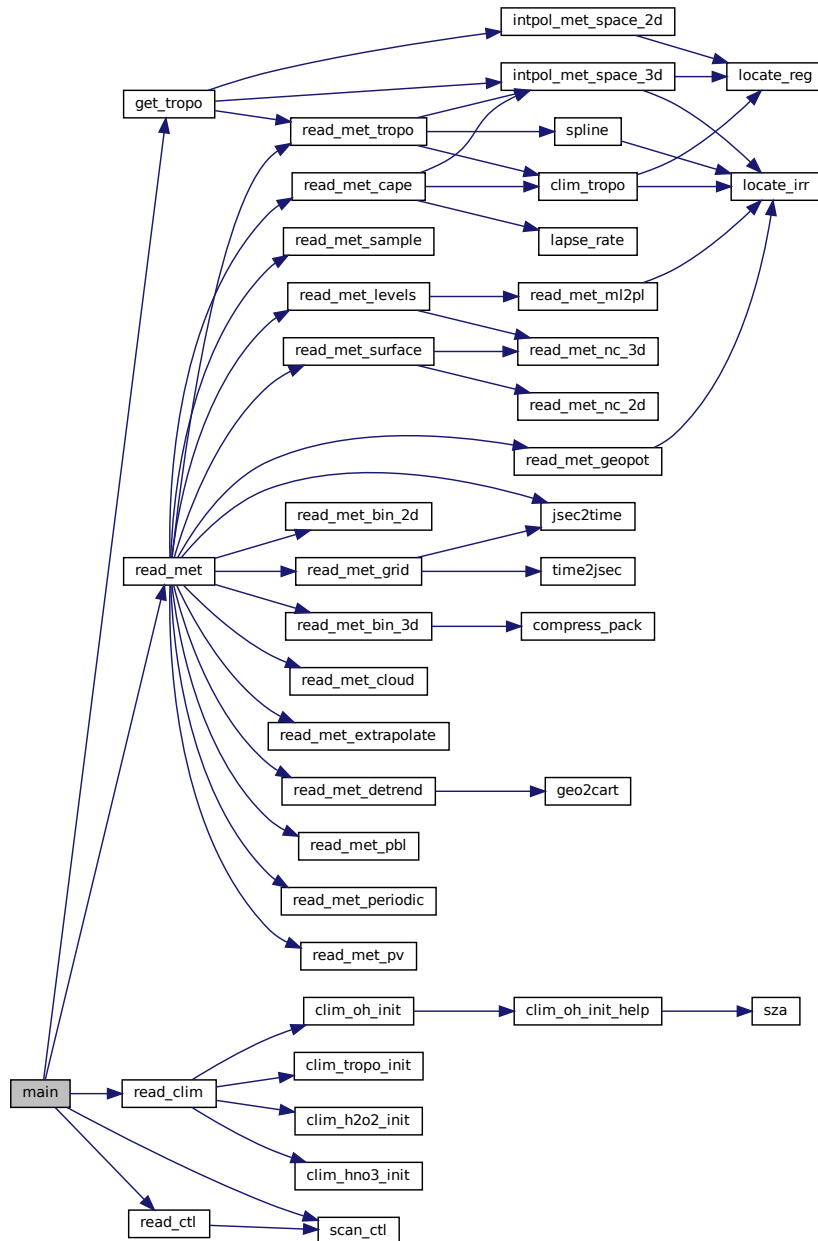
```

```

00233
00234  /* Free... */
00235  free(clim);
00236  free(met);
00237
00238  return EXIT_SUCCESS;
00239 }

```

Here is the call graph for this function:



## 5.48 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 get_tropo(
00032      int met_tropo,
00033      ctl_t *ctl,
00034      clim_t *clim,
00035      met_t *met,
00036      double *lons,
00037      int nx,
00038      double *lats,
00039      int ny,
00040      double *pt,
00041      double *zt,
00042      double *tt,
00043      double *qt);
00044
00045  /* -----
00046   Main...
00047   ----- */
00048
00049  int main(
00050      int argc,
00051      char *argv[]) {
00052
00053      ctl_t ctl;
00054
00055      clim_t *clim;
00056
00057      met_t *met;
00058
00059      static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00060                  lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat;
00061
00062      static int init, i, nx, ny, nt, ncid, varid, dims[3], h2o;
00063
00064      static size_t count[10], start[10];
00065
00066      /* Allocate... */
00067      ALLOC(clim, clim_t, 1);
00068      ALLOC(met, met_t, 1);
00069
00070      /* Check arguments... */
00071      if (argc < 4)
00072          ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00073
00074      /* Read control parameters... */
00075      read_ctl(argv[1], argc, argv, &ctl);
00076      lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00077      lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00078      dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00079      lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00080      lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00081      dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00082      h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00083
00084      /* Read climatological data... */
00085      read_clim(&ctl, clim);
00086
00087      /* Loop over files... */
00088      for (i = 3; i < argc; i++) {
00089
00090          /* Read meteorological data... */
00091          ctl.met_tropo = 0;
00092          if (!read_met(argv[i], &ctl, clim, met))
00093              continue;
00094
00095          /* Set horizontal grid... */
00096          if (!init) {

```

```

00097     init = 1;
00098
00099     /* Get grid... */
00100     if (dlon <= 0)
00101         dlon = fabs(met->lon[1] - met->lon[0]);
00102     if (dlat <= 0)
00103         dlat = fabs(met->lat[1] - met->lat[0]);
00104     if (lon0 < -360 && lon1 > 360) {
00105         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00106         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00107     }
00108     nx = ny = 0;
00109     for (lon = lon0; lon <= lon1; lon += dlon) {
00110         lons[nx] = lon;
00111         if (++nx > EX)
00112             ERRMSG("Too many longitudes!");
00113     }
00114     if (lat0 < -90 && lat1 > 90) {
00115         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00116         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00117     }
00118     for (lat = lat0; lat <= lat1; lat += dlat) {
00119         lats[ny] = lat;
00120         if (++ny > EY)
00121             ERRMSG("Too many latitudes!");
00122     }
00123
00124     /* Create netCDF file... */
00125     LOG(1, "Write tropopause data file: %s", argv[2]);
00126     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00127
00128     /* Create dimensions... */
00129     NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00130     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00131     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00132
00133     /* Create variables... */
00134     NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time",
00135               "seconds since 2000-01-01 00:00:00 UTC");
00136     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[1], "latitude", "degrees_north");
00137     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[2], "longitude", "degrees_east");
00138
00139     NC_DEF_VAR("clp_z", NC_FLOAT, 3, &dims[0], "cold point height", "km");
00140     NC_DEF_VAR("clp_p", NC_FLOAT, 3, &dims[0], "cold point pressure",
00141               "hPa");
00142     NC_DEF_VAR("clp_t", NC_FLOAT, 3, &dims[0], "cold point temperature",
00143               "K");
00144     if (h2o)
00145         NC_DEF_VAR("clp_q", NC_FLOAT, 3, &dims[0], "cold point water vapor",
00146                   "ppv");
00147
00148     NC_DEF_VAR("dyn_z", NC_FLOAT, 3, &dims[0],
00149               "dynamical tropopause height", "km");
00150     NC_DEF_VAR("dyn_p", NC_FLOAT, 3, &dims[0],
00151               "dynamical tropopause pressure", "hPa");
00152     NC_DEF_VAR("dyn_t", NC_FLOAT, 3, &dims[0],
00153               "dynamical tropopause temperature", "K");
00154     if (h2o)
00155         NC_DEF_VAR("dyn_q", NC_FLOAT, 3, &dims[0],
00156                   "dynamical tropopause water vapor", "ppv");
00157
00158     NC_DEF_VAR("wmo_1st_z", NC_FLOAT, 3, &dims[0],
00159               "WMO 1st tropopause height", "km");
00160     NC_DEF_VAR("wmo_1st_p", NC_FLOAT, 3, &dims[0],
00161               "WMO 1st tropopause pressure", "hPa");
00162     NC_DEF_VAR("wmo_1st_t", NC_FLOAT, 3, &dims[0],
00163               "WMO 1st tropopause temperature", "K");
00164     if (h2o)
00165         NC_DEF_VAR("wmo_1st_q", NC_FLOAT, 3, &dims[0],
00166                   "WMO 1st tropopause water vapor", "ppv");
00167
00168     NC_DEF_VAR("wmo_2nd_z", NC_FLOAT, 3, &dims[0],
00169               "WMO 2nd tropopause height", "km");
00170     NC_DEF_VAR("wmo_2nd_p", NC_FLOAT, 3, &dims[0],
00171               "WMO 2nd tropopause pressure", "hPa");
00172     NC_DEF_VAR("wmo_2nd_t", NC_FLOAT, 3, &dims[0],
00173               "WMO 2nd tropopause temperature", "K");
00174     if (h2o)
00175         NC_DEF_VAR("wmo_2nd_q", NC_FLOAT, 3, &dims[0],
00176                   "WMO 2nd tropopause water vapor", "ppv");
00177
00178     /* End definition... */
00179     NC(nc_enddef(ncid));
00180
00181     /* Write longitude and latitude... */
00182     NC_PUT_DOUBLE("lat", lats, 0);
00183     NC_PUT_DOUBLE("lon", lons, 0);

```

```

00184     }
00185
00186     /* Write time... */
00187     start[0] = (size_t) nt;
00188     count[0] = 1;
00189     start[1] = 0;
00190     count[1] = (size_t) ny;
00191     start[2] = 0;
00192     count[2] = (size_t) nx;
00193     NC_PUT_DOUBLE("time", &met->time, 1);
00194
00195     /* Get cold point... */
00196     get_tropo(2, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00197     NC_PUT_DOUBLE("clp_z", zt, 1);
00198     NC_PUT_DOUBLE("clp_p", pt, 1);
00199     NC_PUT_DOUBLE("clp_t", tt, 1);
00200     if (h2o)
00201         NC_PUT_DOUBLE("clp_q", qt, 1);
00202
00203     /* Get dynamical tropopause... */
00204     get_tropo(5, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00205     NC_PUT_DOUBLE("dyn_z", zt, 1);
00206     NC_PUT_DOUBLE("dyn_p", pt, 1);
00207     NC_PUT_DOUBLE("dyn_t", tt, 1);
00208     if (h2o)
00209         NC_PUT_DOUBLE("dyn_q", qt, 1);
00210
00211     /* Get WMO 1st tropopause... */
00212     get_tropo(3, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00213     NC_PUT_DOUBLE("wmo_1st_z", zt, 1);
00214     NC_PUT_DOUBLE("wmo_1st_p", pt, 1);
00215     NC_PUT_DOUBLE("wmo_1st_t", tt, 1);
00216     if (h2o)
00217         NC_PUT_DOUBLE("wmo_1st_q", qt, 1);
00218
00219     /* Get WMO 2nd tropopause... */
00220     get_tropo(4, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt);
00221     NC_PUT_DOUBLE("wmo_2nd_z", zt, 1);
00222     NC_PUT_DOUBLE("wmo_2nd_p", pt, 1);
00223     NC_PUT_DOUBLE("wmo_2nd_t", tt, 1);
00224     if (h2o)
00225         NC_PUT_DOUBLE("wmo_2nd_q", qt, 1);
00226
00227     /* Increment time step counter... */
00228     nt++;
00229 }
00230
00231 /* Close file... */
00232 NC(nc_close(ncid));
00233
00234 /* Free... */
00235 free(clim);
00236 free(met);
00237
00238 return EXIT_SUCCESS;
00239 }
00240
00241 /*****
00242
00243 void get_tropo(
00244     int met_tropo,
00245     ctl_t * ctl,
00246     clim_t * clim,
00247     met_t * met,
00248     double *lons,
00249     int nx,
00250     double *lats,
00251     int ny,
00252     double *pt,
00253     double *zt,
00254     double *tt,
00255     double *qt) {
00256
00257     INTPOL_INIT;
00258
00259     ctl->met_tropo = met_tropo;
00260     read_met_tropo(ctl, clim, met);
00261 #pragma omp parallel for default(shared) private(ci,cw)
00262     for (int ix = 0; ix < nx; ix++)
00263         for (int iy = 0; iy < ny; iy++) {
00264             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00265                                 &pt[iy * nx + ix], ci, cw, 1);
00266             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00267                                 lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00268             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00269                                 lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00270             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],

```

```
00271             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00272     }
00273 }
```

## 5.49 tropo\_sample.c File Reference

Sample tropopause data set.

```
#include "libtrac.h"
```

### Macros

- `#define NT 744`  
*Maximum number of time steps.*

### Functions

- void `intpol_tropo_3d` (double time0, float array0[EX][EY], double time1, float array1[EX][EY], double lons[EX], double lats[EY], int nlon, int 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[ ])

#### 5.49.1 Detailed Description

Sample tropopause data set.

Definition in file [tropo\\_sample.c](#).

#### 5.49.2 Macro Definition Documentation

##### 5.49.2.1 NT `#define NT 744`

Maximum number of time steps.

Definition at line 32 of file [tropo\\_sample.c](#).

#### 5.49.3 Function Documentation

**5.49.3.1 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],
int nlon,
int nlat,
double time,
double lon,
double lat,
int method,
double * var,
double * sigma )

```

3-D linear interpolation of tropopause data.

Definition at line 254 of file tropo\_sample.c.

```

00268     {
00269
00270     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00271
00272     int n = 0;
00273
00274     /* Adjust longitude... */
00275     if (lon < lons[0])
00276         lon += 360;
00277     else if (lon > lons[nlon - 1])
00278         lon -= 360;
00279
00280     /* Get indices... */
00281     int ix = locate_reg(lons, (int) nlon, lon);
00282     int iy = locate_reg(lats, (int) nlat, lat);
00283
00284     /* Calculate standard deviation... */
00285     *sigma = 0;
00286     for (int dx = 0; dx < 2; dx++)
00287         for (int dy = 0; dy < 2; dy++) {
00288             if (isfinite(array0[ix + dx][iy + dy])) {
00289                 mean += array0[ix + dx][iy + dy];
00290                 *sigma += SQR(array0[ix + dx][iy + dy]);
00291                 n++;
00292             }
00293             if (isfinite(array1[ix + dx][iy + dy])) {
00294                 mean += array1[ix + dx][iy + dy];
00295                 *sigma += SQR(array1[ix + dx][iy + dy]);
00296                 n++;
00297             }
00298         }
00299     if (n > 0)
00300         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00301
00302     /* Linear interpolation... */
00303     if (method == 1 && isfinite(array0[ix][iy])
00304         && isfinite(array0[ix][iy + 1])
00305         && isfinite(array0[ix + 1][iy])
00306         && isfinite(array0[ix + 1][iy + 1])
00307         && isfinite(array1[ix][iy])
00308         && isfinite(array1[ix][iy + 1])
00309         && isfinite(array1[ix + 1][iy])
00310         && isfinite(array1[ix + 1][iy + 1])) {
00311
00312         aux00 = LIN(lons[ix], array0[ix][iy],
00313                    lons[ix + 1], array0[ix + 1][iy], lon);
00314         aux01 = LIN(lons[ix], array0[ix][iy + 1],
00315                    lons[ix + 1], array0[ix + 1][iy + 1], lon);
00316         aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00317
00318         aux10 = LIN(lons[ix], array1[ix][iy],
00319                    lons[ix + 1], array1[ix + 1][iy], lon);
00320         aux11 = LIN(lons[ix], array1[ix][iy + 1],
00321                    lons[ix + 1], array1[ix + 1][iy + 1], lon);
00322         aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00323
00324         *var = LIN(time0, aux0, time1, aux1, time);
00325     }
00326

```

```

00327  /* Nearest neighbor interpolation... */
00328  else {
00329      aux00 = NN(lons[ix], array0[ix][iy],
00330                lons[ix + 1], array0[ix + 1][iy], lon);
00331      aux01 = NN(lons[ix], array0[ix][iy + 1],
00332                lons[ix + 1], array0[ix + 1][iy + 1], lon);
00333      aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00334
00335      aux10 = NN(lons[ix], array1[ix][iy],
00336                lons[ix + 1], array1[ix + 1][iy], lon);
00337      aux11 = NN(lons[ix], array1[ix][iy + 1],
00338                lons[ix + 1], array1[ix + 1][iy + 1], lon);
00339      aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00340
00341      *var = NN(time0, aux0, time1, aux1, time);
00342  }
00343 }

```

Here is the call graph for this function:



```

5.49.3.2 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, ncid, varid, varid_z,
00079                  varid_p, varid_t, varid_q, h2o, ntime, nlon, nlat, ilon, ilat;
00080
00081      static size_t count[10], start[10];
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     LOG(1, "Read tropopause data: %s", argv[3]);
00101     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00102         ERRMSG("Cannot open file!");
00103
00104     /* Get dimensions... */
00105     NC_INQ_DIM("time", &ntime, 1, NT);
00106     NC_INQ_DIM("lat", &nlat, 1, EY);
00107     NC_INQ_DIM("lon", &nlon, 1, EX);
00108
00109     /* Read coordinates... */
00110     NC_GET_DOUBLE("time", times, 1);
00111     NC_GET_DOUBLE("lat", lats, 1);
00112     NC_GET_DOUBLE("lon", lons, 1);
00113
00114     /* Get variable indices... */
00115     sprintf(varname, "%s_z", argv[4]);
00116     NC(nc_inq_varid(ncid, varname, &varid_z));
00117     sprintf(varname, "%s_p", argv[4]);
00118     NC(nc_inq_varid(ncid, varname, &varid_p));
00119     sprintf(varname, "%s_t", argv[4]);
00120     NC(nc_inq_varid(ncid, varname, &varid_t));
00121     sprintf(varname, "%s_q", argv[4]);
00122     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00123
00124     /* Set dimensions... */
00125     count[0] = 1;
00126     count[1] = (size_t) nlat;
00127     count[2] = (size_t) nlon;
00128
00129     /* Create file... */
00130     LOG(1, "Write tropopause sample data: %s", argv[2]);
00131     if (!(out = fopen(argv[2], "w")))
00132         ERRMSG("Cannot create file!");
00133
00134     /* Write header... */
00135     fprintf(out,
00136         "# $1 = time [s]\n"
00137         "# $2 = altitude [km]\n"
00138         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00139     for (iq = 0; iq < ctl.nq; iq++)
00140         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00141             ctl.qnt_unit[iq]);
00142     fprintf(out, "# $%d = tropopause height [km]\n", 5 + ctl.nq);
00143     fprintf(out, "# $%d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00144     fprintf(out, "# $%d = tropopause temperature [K]\n", 7 + ctl.nq);
00145     fprintf(out, "# $%d = tropopause water vapor [ppv]\n", 8 + ctl.nq);
00146     fprintf(out, "# $%d = tropopause height (sigma) [km]\n", 9 + ctl.nq);
00147     fprintf(out, "# $%d = tropopause pressure (sigma) [hPa]\n", 10 + ctl.nq);
00148     fprintf(out, "# $%d = tropopause temperature (sigma) [K]\n", 11 + ctl.nq);
00149     fprintf(out, "# $%d = tropopause water vapor (sigma) [ppv]\n",
00150         12 + ctl.nq);
00151
00152     /* Loop over particles... */
00153     for (ip = 0; ip < atm->np; ip++) {
00154
00155         /* Check temporal ordering... */
00156         if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00157             ERRMSG("Time must be ascending!");
00158
00159         /* Check range... */
00160         if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00161             continue;
00162
00163         /* Read data... */
00164         it = locate_irr(times, (int) ntime, atm->time[ip]);
00165         if (it != it_old) {
00166
00167             time0 = times[it];
00168             start[0] = (size_t) it;
00169             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00170             for (ilon = 0; ilon < nlon; ilon++)
00171                 for (ilat = 0; ilat < nlat; ilat++)
00172                     tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00173             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00174             for (ilon = 0; ilon < nlon; ilon++)
00175                 for (ilat = 0; ilat < nlat; ilat++)
00176                     tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00177             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00178             for (ilon = 0; ilon < nlon; ilon++)
00179                 for (ilat = 0; ilat < nlat; ilat++)
00180                     tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00181             if (h2o) {
00182                 NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00183                 for (ilon = 0; ilon < nlon; ilon++)

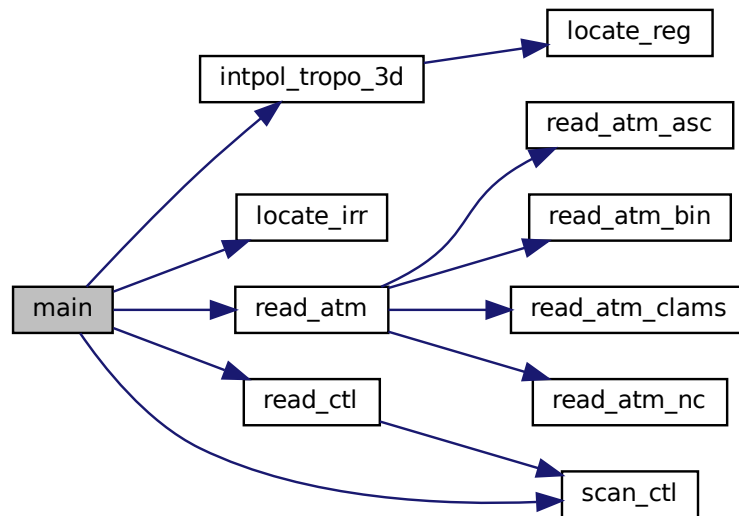
```

```

00184         for (ilat = 0; ilat < nlat; ilat++)
00185             tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00186     } else
00187         for (ilon = 0; ilon < nlon; ilon++)
00188             for (ilat = 0; ilat < nlat; ilat++)
00189                 tropo_q0[ilon][ilat] = GSL_NAN;
00190
00191     time1 = times[it + 1];
00192     start[0] = (size_t) it + 1;
00193     NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00194     for (ilon = 0; ilon < nlon; ilon++)
00195         for (ilat = 0; ilat < nlat; ilat++)
00196             tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00197     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00198     for (ilon = 0; ilon < nlon; ilon++)
00199         for (ilat = 0; ilat < nlat; ilat++)
00200             tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00201     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00202     for (ilon = 0; ilon < nlon; ilon++)
00203         for (ilat = 0; ilat < nlat; ilat++)
00204             tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00205     if (h2o) {
00206         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00207         for (ilon = 0; ilon < nlon; ilon++)
00208             for (ilat = 0; ilat < nlat; ilat++)
00209                 tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00210     } else
00211         for (ilon = 0; ilon < nlon; ilon++)
00212             for (ilat = 0; ilat < nlat; ilat++)
00213                 tropo_q1[ilon][ilat] = GSL_NAN;;
00214 }
00215 it_old = it;
00216
00217 /* Interpolate... */
00218 intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00219                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00220                atm->lat[ip], method, &z0, &z0sig);
00221 intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00222                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00223                atm->lat[ip], method, &p0, &p0sig);
00224 intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00225                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00226                atm->lat[ip], method, &t0, &t0sig);
00227 intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00228                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00229                atm->lat[ip], method, &q0, &q0sig);
00230
00231 /* Write output... */
00232 fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00233         atm->lon[ip], atm->lat[ip]);
00234 for (iq = 0; iq < ctl.nq; iq++) {
00235     fprintf(out, " ");
00236     fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00237 }
00238 fprintf(out, " %g %g %g %g %g %g %g %g\n",
00239         z0, p0, t0, q0, z0sig, p0sig, t0sig, q0sig);
00240 }
00241
00242 /* Close files... */
00243 fclose(out);
00244 NC(nc_close(ncid));
00245
00246 /* Free... */
00247 free(atm);
00248
00249 return EXIT_SUCCESS;
00250 }

```

Here is the call graph for this function:



## 5.50 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     int nlon,
00047     int 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
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, ncid, varid, varid_z,
00079     varid_p, varid_t, varid_q, h2o, ntime, nlon, nlat, ilon, ilat;
00080
00081     static size_t count[10], start[10];
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     LOG(1, "Read tropopause data: %s", argv[3]);
00101     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00102         ERRMSG("Cannot open file!");
00103
00104     /* Get dimensions... */
00105     NC_INQ_DIM("time", &ntime, 1, NT);
00106     NC_INQ_DIM("lat", &nlat, 1, EY);
00107     NC_INQ_DIM("lon", &nlon, 1, EX);
00108
00109     /* Read coordinates... */
00110     NC_GET_DOUBLE("time", times, 1);
00111     NC_GET_DOUBLE("lat", lats, 1);
00112     NC_GET_DOUBLE("lon", lons, 1);
00113
00114     /* Get variable indices... */
00115     sprintf(varname, "%s_z", argv[4]);
00116     NC(nc_inq_varid(ncid, varname, &varid_z));
00117     sprintf(varname, "%s_p", argv[4]);
00118     NC(nc_inq_varid(ncid, varname, &varid_p));
00119     sprintf(varname, "%s_t", argv[4]);
00120     NC(nc_inq_varid(ncid, varname, &varid_t));
00121     sprintf(varname, "%s_q", argv[4]);
00122     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00123
00124     /* Set dimensions... */
00125     count[0] = 1;
00126     count[1] = (size_t) nlat;
00127     count[2] = (size_t) nlon;
00128
00129     /* Create file... */
00130     LOG(1, "Write tropopause sample data: %s", argv[2]);
00131     if (!(out = fopen(argv[2], "w")))
00132         ERRMSG("Cannot create file!");
00133
00134     /* Write header... */
00135     fprintf(out,
00136         "# $1 = time [s]\n"
00137         "# $2 = altitude [km]\n"
00138         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");

```

```

00139     for (iq = 0; iq < ctl.nq; iq++)
00140         fprintf(out, "# %i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00141             ctl.qnt_unit[iq]);
00142     fprintf(out, "# %d = tropopause height [km]\n", 5 + ctl.nq);
00143     fprintf(out, "# %d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00144     fprintf(out, "# %d = tropopause temperature [K]\n", 7 + ctl.nq);
00145     fprintf(out, "# %d = tropopause water vapor [ppv]\n", 8 + ctl.nq);
00146     fprintf(out, "# %d = tropopause height (sigma) [km]\n", 9 + ctl.nq);
00147     fprintf(out, "# %d = tropopause pressure (sigma) [hPa]\n", 10 + ctl.nq);
00148     fprintf(out, "# %d = tropopause temperature (sigma) [K]\n", 11 + ctl.nq);
00149     fprintf(out, "# %d = tropopause water vapor (sigma) [ppv]\n",
00150         12 + ctl.nq);
00151
00152     /* Loop over particles... */
00153     for (ip = 0; ip < atm->np; ip++) {
00154
00155         /* Check temporal ordering... */
00156         if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00157             ERRMSG("Time must be ascending!");
00158
00159         /* Check range... */
00160         if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00161             continue;
00162
00163         /* Read data... */
00164         it = locate_irr(times, (int) ntime, atm->time[ip]);
00165         if (it != it_old) {
00166
00167             time0 = times[it];
00168             start[0] = (size_t) it;
00169             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00170             for (ilon = 0; ilon < nlon; ilon++)
00171                 for (ilat = 0; ilat < nlat; ilat++)
00172                     tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00173             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00174             for (ilon = 0; ilon < nlon; ilon++)
00175                 for (ilat = 0; ilat < nlat; ilat++)
00176                     tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00177             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00178             for (ilon = 0; ilon < nlon; ilon++)
00179                 for (ilat = 0; ilat < nlat; ilat++)
00180                     tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00181             if (h2o) {
00182                 NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00183                 for (ilon = 0; ilon < nlon; ilon++)
00184                     for (ilat = 0; ilat < nlat; ilat++)
00185                         tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00186             } else
00187                 for (ilon = 0; ilon < nlon; ilon++)
00188                     for (ilat = 0; ilat < nlat; ilat++)
00189                         tropo_q0[ilon][ilat] = GSL_NAN;
00190
00191             time1 = times[it + 1];
00192             start[0] = (size_t) it + 1;
00193             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00194             for (ilon = 0; ilon < nlon; ilon++)
00195                 for (ilat = 0; ilat < nlat; ilat++)
00196                     tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00197             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00198             for (ilon = 0; ilon < nlon; ilon++)
00199                 for (ilat = 0; ilat < nlat; ilat++)
00200                     tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00201             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00202             for (ilon = 0; ilon < nlon; ilon++)
00203                 for (ilat = 0; ilat < nlat; ilat++)
00204                     tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00205             if (h2o) {
00206                 NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00207                 for (ilon = 0; ilon < nlon; ilon++)
00208                     for (ilat = 0; ilat < nlat; ilat++)
00209                         tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00210             } else
00211                 for (ilon = 0; ilon < nlon; ilon++)
00212                     for (ilat = 0; ilat < nlat; ilat++)
00213                         tropo_q1[ilon][ilat] = GSL_NAN;
00214             }
00215             it_old = it;
00216
00217         /* Interpolate... */
00218         intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00219             lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00220             atm->lat[ip], method, &z0, &z0sig);
00221         intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00222             lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00223             atm->lat[ip], method, &p0, &p0sig);
00224         intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00225             lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],

```

```

00226         atm->lat[ip], method, &t0, &t0sig);
00227     intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00228         lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00229         atm->lat[ip], method, &q0, &q0sig);
00230
00231     /* Write output... */
00232     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00233         atm->lon[ip], atm->lat[ip]);
00234     for (iq = 0; iq < ctl.nq; iq++) {
00235         fprintf(out, " ");
00236         fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00237     }
00238     fprintf(out, " %g %g %g %g %g %g %g %g\n",
00239         z0, p0, t0, q0, z0sig, p0sig, t0sig, q0sig);
00240 }
00241
00242 /* Close files... */
00243 fclose(out);
00244 NC(nc_close(ncid));
00245
00246 /* Free... */
00247 free(atm);
00248
00249 return EXIT_SUCCESS;
00250 }
00251
00252 /*****
00253
00254 void intpol_tropo_3d(
00255     double time0,
00256     float array0[EX][EY],
00257     double time1,
00258     float array1[EX][EY],
00259     double lons[EX],
00260     double lats[EY],
00261     int nlon,
00262     int nlat,
00263     double time,
00264     double lon,
00265     double lat,
00266     int method,
00267     double *var,
00268     double *sigma) {
00269
00270     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00271
00272     int n = 0;
00273
00274     /* Adjust longitude... */
00275     if (lon < lons[0])
00276         lon += 360;
00277     else if (lon > lons[nlon - 1])
00278         lon -= 360;
00279
00280     /* Get indices... */
00281     int ix = locate_reg(lons, (int) nlon, lon);
00282     int iy = locate_reg(lats, (int) nlat, lat);
00283
00284     /* Calculate standard deviation... */
00285     *sigma = 0;
00286     for (int dx = 0; dx < 2; dx++)
00287         for (int dy = 0; dy < 2; dy++) {
00288             if (isfinite(array0[ix + dx][iy + dy])) {
00289                 mean += array0[ix + dx][iy + dy];
00290                 *sigma += SQR(array0[ix + dx][iy + dy]);
00291                 n++;
00292             }
00293             if (isfinite(array1[ix + dx][iy + dy])) {
00294                 mean += array1[ix + dx][iy + dy];
00295                 *sigma += SQR(array1[ix + dx][iy + dy]);
00296                 n++;
00297             }
00298         }
00299     if (n > 0)
00300         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00301
00302     /* Linear interpolation... */
00303     if (method == 1 && isfinite(array0[ix][iy])
00304         && isfinite(array0[ix][iy + 1])
00305         && isfinite(array0[ix + 1][iy])
00306         && isfinite(array0[ix + 1][iy + 1])
00307         && isfinite(array1[ix][iy])
00308         && isfinite(array1[ix][iy + 1])
00309         && isfinite(array1[ix + 1][iy])
00310         && isfinite(array1[ix + 1][iy + 1])) {
00311
00312         aux00 = LIN(lons[ix], array0[ix][iy],

```

```

00313         lons[ix + 1], array0[ix + 1][iy], lon);
00314     aux01 = LIN(lons[ix], array0[ix][iy + 1],
00315         lons[ix + 1], array0[ix + 1][iy + 1], lon);
00316     aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00317
00318     aux10 = LIN(lons[ix], array1[ix][iy],
00319         lons[ix + 1], array1[ix + 1][iy], lon);
00320     aux11 = LIN(lons[ix], array1[ix][iy + 1],
00321         lons[ix + 1], array1[ix + 1][iy + 1], lon);
00322     aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00323
00324     *var = LIN(time0, aux0, time1, aux1, time);
00325 }
00326
00327 /* Nearest neighbor interpolation... */
00328 else {
00329     aux00 = NN(lons[ix], array0[ix][iy],
00330         lons[ix + 1], array0[ix + 1][iy], lon);
00331     aux01 = NN(lons[ix], array0[ix][iy + 1],
00332         lons[ix + 1], array0[ix + 1][iy + 1], lon);
00333     aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00334
00335     aux10 = NN(lons[ix], array1[ix][iy],
00336         lons[ix + 1], array1[ix + 1][iy], lon);
00337     aux11 = NN(lons[ix], array1[ix][iy + 1],
00338         lons[ix + 1], array1[ix + 1][iy + 1], lon);
00339     aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00340
00341     *var = NN(time0, aux0, time1, aux1, time);
00342 }
00343 }

```

## 5.51 tropo\_zm.c File Reference

Extract zonal mean of tropopause data set.

```
#include "libtrac.h"
```

### Macros

- `#define NT 744`  
*Maximum number of time steps.*

### Functions

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

#### 5.51.1 Detailed Description

Extract zonal mean of tropopause data set.

Definition in file [tropo\\_zm.c](#).

#### 5.51.2 Macro Definition Documentation

### 5.51.2.1 NT #define NT 744

Maximum number of time steps.

Definition at line 32 of file [tropo\\_zm.c](#).

## 5.51.3 Function Documentation

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

Definition at line 38 of file [tropo\\_zm.c](#).

```

00040     {
00041
00042     ctl_t ctl;
00043
00044     static FILE *out;
00045
00046     static char tstr[LEN], varname[LEN];
00047
00048     static double time0, lons[EX], lats[EY], zm[EY], zs[EY], pm[EY],
00049         ps[EY], tm[EY], ts[EY], qm[EY], qs[EY];
00050
00051     static float help[EX * EY], tropo_z0[EX][EY], tropo_p0[EX][EY],
00052         tropo_t0[EX][EY], tropo_q0[EX][EY];
00053
00054     static int ncid, varid, varid_z, varid_p, varid_t, varid_q, h2o,
00055         n[EY], nt[EY], year, mon, day, init, ntime, nlon, nlat, ilon, ilat;
00056
00057     static size_t count[10], start[10];
00058
00059     /* Check arguments... */
00060     if (argc < 5)
00061         ERRMSG("Give parameters: <ctl> <zm.tab> <var> <tropo.nc>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065
00066     /* Loop over tropopause files... */
00067     for (int iarg = 4; iarg < argc; iarg++) {
00068
00069         /* Open tropopause file... */
00070         LOG(1, "Read tropopause data: %s", argv[iarg]);
00071         if (nc_open(argv[iarg], NC_NOWRITE, &ncid) != NC_NOERR)
00072             ERRMSG("Cannot open file!");
00073
00074         /* Get dimensions... */
00075         NC_INQ_DIM("time", &ntime, 1, NT);
00076         NC_INQ_DIM("lat", &nlat, 1, EY);
00077         NC_INQ_DIM("lon", &nlon, 1, EX);
00078
00079         /* Read coordinates... */
00080         NC_GET_DOUBLE("lat", lats, 1);
00081         NC_GET_DOUBLE("lon", lons, 1);
00082
00083         /* Get variable indices... */
00084         sprintf(varname, "%s_z", argv[3]);
00085         NC(nc_inq_varid(ncid, varname, &varid_z));
00086         sprintf(varname, "%s_p", argv[3]);
00087         NC(nc_inq_varid(ncid, varname, &varid_p));
00088         sprintf(varname, "%s_t", argv[3]);
00089         NC(nc_inq_varid(ncid, varname, &varid_t));
00090         sprintf(varname, "%s_q", argv[3]);
00091         h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00092
00093         /* Set dimensions... */
00094         count[0] = 1;
00095         count[1] = (size_t) nlat;
00096         count[2] = (size_t) nlon;
00097
00098         /* Loop over time steps... */
00099         for (int it = 0; it < ntime; it++) {
00100

```



```

00101      /* Get time from filename... */
00102      if (!init) {
00103          init = 1;
00104          size_t len = strlen(argv[iarg]);
00105          sprintf(tstr, "%.4s", &argv[iarg][len - 13]);
00106          year = atoi(tstr);
00107          sprintf(tstr, "%.2s", &argv[iarg][len - 8]);
00108          mon = atoi(tstr);
00109          sprintf(tstr, "%.2s", &argv[iarg][len - 5]);
00110          day = atoi(tstr);
00111          time2jsec(year, mon, day, 0, 0, 0, 0, &time0);
00112      }
00113
00114      /* Read data... */
00115      start[0] = (size_t) it;
00116      NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00117      for (ilon = 0; ilon < nlon; ilon++)
00118          for (ilat = 0; ilat < nlat; ilat++)
00119              tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00120      NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00121      for (ilon = 0; ilon < nlon; ilon++)
00122          for (ilat = 0; ilat < nlat; ilat++)
00123              tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00124      NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00125      for (ilon = 0; ilon < nlon; ilon++)
00126          for (ilat = 0; ilat < nlat; ilat++)
00127              tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00128      if (h2o) {
00129          NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00130          for (ilon = 0; ilon < nlon; ilon++)
00131              for (ilat = 0; ilat < nlat; ilat++)
00132                  tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00133      } else
00134          for (ilon = 0; ilon < nlon; ilon++)
00135              for (ilat = 0; ilat < nlat; ilat++)
00136                  tropo_q0[ilon][ilat] = GSL_NAN;
00137
00138      /* Averaging... */
00139      for (ilat = 0; ilat < nlat; ilat++)
00140          for (ilon = 0; ilon < nlon; ilon++) {
00141              nt[ilat]++;
00142              if (isfinite(tropo_z0[ilon][ilat])
00143                  && isfinite(tropo_p0[ilon][ilat])
00144                  && isfinite(tropo_t0[ilon][ilat])
00145                  && (!h2o || isfinite(tropo_q0[ilon][ilat]))) {
00146                  zm[ilat] += tropo_z0[ilon][ilat];
00147                  zs[ilat] += SQR(tropo_z0[ilon][ilat]);
00148                  pm[ilat] += tropo_p0[ilon][ilat];
00149                  ps[ilat] += SQR(tropo_p0[ilon][ilat]);
00150                  tm[ilat] += tropo_t0[ilon][ilat];
00151                  ts[ilat] += SQR(tropo_t0[ilon][ilat]);
00152                  qm[ilat] += tropo_q0[ilon][ilat];
00153                  qs[ilat] += SQR(tropo_q0[ilon][ilat]);
00154                  n[ilat]++;
00155              }
00156          }
00157
00158      /* Close files... */
00159      NC(nc_close(ncid));
00160  }
00161
00162      /* Normalize... */
00163      for (ilat = 0; ilat < nlat; ilat++)
00164          if (n[ilat] > 0) {
00165              zm[ilat] /= n[ilat];
00166              pm[ilat] /= n[ilat];
00167              tm[ilat] /= n[ilat];
00168              qm[ilat] /= n[ilat];
00169              double aux = zs[ilat] / n[ilat] - SQR(zm[ilat]);
00170              zs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00171              aux = ps[ilat] / n[ilat] - SQR(pm[ilat]);
00172              ps[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00173              aux = ts[ilat] / n[ilat] - SQR(tm[ilat]);
00174              ts[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00175              aux = qs[ilat] / n[ilat] - SQR(qm[ilat]);
00176              qs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00177          }
00178
00179      /* Create file... */
00180      LOG(1, "Write tropopause zonal mean data: %s", argv[2]);
00181      if (!(out = fopen(argv[2], "w")))
00182          ERRMSG("Cannot create file!");
00183
00184      /* Write header... */
00185      fprintf(out,
00186          "# $1 = time [s]\n"

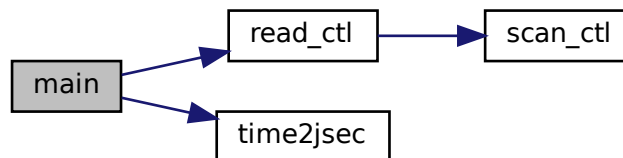
```

```

00188     "# $2 = latitude [deg]\n"
00189     "# $3 = tropopause height (mean) [km]\n"
00190     "# $4 = tropopause pressure (mean) [hPa]\n"
00191     "# $5 = tropopause temperature (mean) [K]\n"
00192     "# $6 = tropopause water vapor (mean) [ppv]\n"
00193     "# $7 = tropopause height (sigma) [km]\n"
00194     "# $8 = tropopause pressure (sigma) [hPa]\n"
00195     "# $9 = tropopause temperature (sigma) [K]\n"
00196     "# $10 = tropopause water vapor (sigma) [ppv]\n"
00197     "# $11 = number of data points\n"
00198     "# $12 = occurrence frequency [%]\n\n");
00199
00200 /* Write output... */
00201 for (ilat = 0; ilat < nlat; ilat++)
00202     fprintf(out, "%.2f %g %g %g %g %g %g %g %d %g\n", time0, lats[ilat],
00203             zm[ilat], pm[ilat], tm[ilat], qm[ilat], zs[ilat], ps[ilat],
00204             ts[ilat], qs[ilat], n[ilat], 100. * n[ilat] / nt[ilat]);
00205
00206 /* Close files... */
00207 fclose(out);
00208
00209 return EXIT_SUCCESS;
00210 }

```

Here is the call graph for this function:



## 5.52 tropo\_zm.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2022 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 /* -----
00023  Dimensions...
00024  ----- */
00025
00026 #define NT 744
00027
00028 /* -----
00029  Main...
00030  ----- */
00031
00032 int main(
00033     int argc,
00034     char *argv[]) {
00035
00036

```

```

00042     ctl_t ctl;
00043
00044     static FILE *out;
00045
00046     static char tstr[LEN], varname[LEN];
00047
00048     static double time0, lons[EX], lats[EY], zm[EY], zs[EY], pm[EY],
00049         ps[EY], tm[EY], ts[EY], qm[EY], qs[EY];
00050
00051     static float help[EX * EY], tropo_z0[EX][EY], tropo_p0[EX][EY],
00052         tropo_t0[EX][EY], tropo_q0[EX][EY];
00053
00054     static int ncid, varid, varid_z, varid_p, varid_t, varid_q, h2o,
00055         n[EY], nt[EY], year, mon, day, init, ntime, nlon, nlat, ilon, ilat;
00056
00057     static size_t count[10], start[10];
00058
00059     /* Check arguments... */
00060     if (argc < 5)
00061         ERRMSG("Give parameters: <ctl> <zm.tab> <var> <tropo.nc>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065
00066     /* Loop over tropopause files... */
00067     for (int iarg = 4; iarg < argc; iarg++) {
00068
00069         /* Open tropopause file... */
00070         LOG(1, "Read tropopause data: %s", argv[iarg]);
00071         if (nc_open(argv[iarg], NC_NOWRITE, &ncid) != NC_NOERR)
00072             ERRMSG("Cannot open file!");
00073
00074         /* Get dimensions... */
00075         NC_INQ_DIM("time", &ntime, 1, NT);
00076         NC_INQ_DIM("lat", &nlat, 1, EY);
00077         NC_INQ_DIM("lon", &nlon, 1, EX);
00078
00079         /* Read coordinates... */
00080         NC_GET_DOUBLE("lat", lats, 1);
00081         NC_GET_DOUBLE("lon", lons, 1);
00082
00083         /* Get variable indices... */
00084         sprintf(varname, "%s_z", argv[3]);
00085         NC(nc_inq_varid(ncid, varname, &varid_z));
00086         sprintf(varname, "%s_p", argv[3]);
00087         NC(nc_inq_varid(ncid, varname, &varid_p));
00088         sprintf(varname, "%s_t", argv[3]);
00089         NC(nc_inq_varid(ncid, varname, &varid_t));
00090         sprintf(varname, "%s_q", argv[3]);
00091         h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00092
00093         /* Set dimensions... */
00094         count[0] = 1;
00095         count[1] = (size_t) nlat;
00096         count[2] = (size_t) nlon;
00097
00098         /* Loop over time steps... */
00099         for (int it = 0; it < ntime; it++) {
00100
00101             /* Get time from filename... */
00102             if (!init) {
00103                 init = 1;
00104                 size_t len = strlen(argv[iarg]);
00105                 sprintf(tstr, "%.4s", &argv[iarg][len - 13]);
00106                 year = atoi(tstr);
00107                 sprintf(tstr, "%.2s", &argv[iarg][len - 8]);
00108                 mon = atoi(tstr);
00109                 sprintf(tstr, "%.2s", &argv[iarg][len - 5]);
00110                 day = atoi(tstr);
00111                 time2jsec(year, mon, day, 0, 0, 0, 0, &time0);
00112             }
00113
00114             /* Read data... */
00115             start[0] = (size_t) it;
00116             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00117             for (ilon = 0; ilon < nlon; ilon++)
00118                 for (ilat = 0; ilat < nlat; ilat++)
00119                     tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00120             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00121             for (ilon = 0; ilon < nlon; ilon++)
00122                 for (ilat = 0; ilat < nlat; ilat++)
00123                     tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00124             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00125             for (ilon = 0; ilon < nlon; ilon++)
00126                 for (ilat = 0; ilat < nlat; ilat++)
00127                     tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00128             if (h2o) {

```

```

00129         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00130     for (ilon = 0; ilon < nlon; ilon++)
00131         for (ilat = 0; ilat < nlat; ilat++)
00132             tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00133     } else
00134         for (ilon = 0; ilon < nlon; ilon++)
00135             for (ilat = 0; ilat < nlat; ilat++)
00136                 tropo_q0[ilon][ilat] = GSL_NAN;
00137
00138     /* Averaging... */
00139     for (ilat = 0; ilat < nlat; ilat++)
00140         for (ilon = 0; ilon < nlon; ilon++) {
00141             nt[ilat]++;
00142             if (isfinite(tropo_z0[ilon][ilat])
00143                 && isfinite(tropo_p0[ilon][ilat])
00144                 && isfinite(tropo_t0[ilon][ilat])
00145                 && (!h2o || isfinite(tropo_q0[ilon][ilat]))) {
00146                 zm[ilat] += tropo_z0[ilon][ilat];
00147                 zs[ilat] += SQR(tropo_z0[ilon][ilat]);
00148                 pm[ilat] += tropo_p0[ilon][ilat];
00149                 ps[ilat] += SQR(tropo_p0[ilon][ilat]);
00150                 tm[ilat] += tropo_t0[ilon][ilat];
00151                 ts[ilat] += SQR(tropo_t0[ilon][ilat]);
00152                 qm[ilat] += tropo_q0[ilon][ilat];
00153                 qs[ilat] += SQR(tropo_q0[ilon][ilat]);
00154                 n[ilat]++;
00155             }
00156         }
00157     }
00158
00159     /* Close files... */
00160     NC(nc_close(ncid));
00161 }
00162
00163 /* Normalize... */
00164 for (ilat = 0; ilat < nlat; ilat++)
00165     if (n[ilat] > 0) {
00166         zm[ilat] /= n[ilat];
00167         pm[ilat] /= n[ilat];
00168         tm[ilat] /= n[ilat];
00169         qm[ilat] /= n[ilat];
00170         double aux = zs[ilat] / n[ilat] - SQR(zm[ilat]);
00171         zs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00172         aux = ps[ilat] / n[ilat] - SQR(pm[ilat]);
00173         ps[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00174         aux = ts[ilat] / n[ilat] - SQR(tm[ilat]);
00175         ts[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00176         aux = qs[ilat] / n[ilat] - SQR(qm[ilat]);
00177         qs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00178     }
00179
00180 /* Create file... */
00181 LOG(1, "Write tropopause zonal mean data: %s", argv[2]);
00182 if (!(out = fopen(argv[2], "w")))
00183     ERRMSG("Cannot create file!");
00184
00185 /* Write header... */
00186 fprintf(out,
00187     "# $1 = time [s]\n"
00188     "# $2 = latitude [deg]\n"
00189     "# $3 = tropopause height (mean) [km]\n"
00190     "# $4 = tropopause pressure (mean) [hPa]\n"
00191     "# $5 = tropopause temperature (mean) [K]\n"
00192     "# $6 = tropopause water vapor (mean) [ppv]\n"
00193     "# $7 = tropopause height (sigma) [km]\n"
00194     "# $8 = tropopause pressure (sigma) [hPa]\n"
00195     "# $9 = tropopause temperature (sigma) [K]\n"
00196     "# $10 = tropopause water vapor (sigma) [ppv]\n"
00197     "# $11 = number of data points\n"
00198     "# $12 = occurrence frequency [%]\n\n");
00199
00200 /* Write output... */
00201 for (ilat = 0; ilat < nlat; ilat++)
00202     fprintf(out, "%.2f %g %g %g %g %g %g %g %d %g\n", time0, lats[ilat],
00203         zm[ilat], pm[ilat], tm[ilat], qm[ilat], zs[ilat], ps[ilat],
00204         ts[ilat], qs[ilat], n[ilat], 100. * n[ilat] / nt[ilat]);
00205
00206 /* Close files... */
00207 fclose(out);
00208
00209 return EXIT_SUCCESS;
00210 }

```

## 5.53 wind.c File Reference

Create meteorological data files with synthetic wind fields.

```
#include "libtrac.h"
```

### Functions

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

#### 5.53.1 Detailed Description

Create meteorological data files with synthetic wind fields.

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

#### 5.53.2 Function Documentation

**5.53.2.1 main()** int main (

```
int argc,
char * argv[] )
```

Definition at line 31 of file [wind.c](#).

```
00033     {
00034
00035     ctl_t ctl;
00036
00037     static char filename[LEN];
00038
00039     static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00040         u0, u1, w0, alpha;
00041
00042     static float *dataT, *dataU, *dataV, *dataW;
00043
00044     static int ncid, varid, dims[4], idx, ix, iy, iz, nx, ny, nz,
00045         year, mon, day, hour, min, sec;
00046
00047     static size_t start[4], count[4];
00048
00049     /* Allocate... */
00050     ALLOC(dataT, float,
00051         EP * EY * EX);
00052     ALLOC(dataU, float,
00053         EP * EY * EX);
00054     ALLOC(dataV, float,
00055         EP * EY * EX);
00056     ALLOC(dataW, float,
00057         EP * EY * EX);
00058
00059     /* Check arguments... */
00060     if (argc < 3)
00061         ERRMSG("Give parameters: <ctl> <metbase>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065     t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00066     nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00067     ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00068     nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00069     z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00070     z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00071     u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
```

```

00072 u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00073 w0 = scan_ctl(argv[1], argc, argv, "WIND_W0", -1, "0", NULL);
00074 alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00075
00076 /* Check dimensions... */
00077 if (nx < 1 || nx > EX)
00078     ERRMSG("Set 1 <= NX <= MAX!");
00079 if (ny < 1 || ny > EY)
00080     ERRMSG("Set 1 <= NY <= MAX!");
00081 if (nz < 1 || nz > EP)
00082     ERRMSG("Set 1 <= NZ <= MAX!");
00083
00084 /* Get time... */
00085 jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00086 t0 = year * 10000. + mon * 100. + day + hour / 24.;
00087
00088 /* Set filename... */
00089 sprintf(filename, "%s_d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00090
00091 /* Create netCDF file... */
00092 NC(nc_create(filename, NC_CLOBBER, &ncid));
00093
00094 /* Create dimensions... */
00095 NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00096 NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00097 NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00098 NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00099
00100 /* Create variables... */
00101 NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time", "day as %Y%m%d.%f");
00102 NC_DEF_VAR("lev", NC_DOUBLE, 1, &dims[1], "air_pressure", "Pa");
00103 NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[2], "latitude", "degrees_north");
00104 NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[3], "longitude", "degrees_east");
00105 NC_DEF_VAR("T", NC_FLOAT, 4, &dims[0], "Temperature", "K");
00106 NC_DEF_VAR("U", NC_FLOAT, 4, &dims[0], "zonal wind", "m s**-1");
00107 NC_DEF_VAR("V", NC_FLOAT, 4, &dims[0], "meridional wind", "m s**-1");
00108 NC_DEF_VAR("W", NC_FLOAT, 4, &dims[0], "vertical velocity", "Pa s**-1");
00109
00110 /* End definition... */
00111 NC(nc_enddef(ncid));
00112
00113 /* Set coordinates... */
00114 for (ix = 0; ix < nx; ix++)
00115     dataLon[ix] = 360.0 / nx * (double) ix;
00116 for (iy = 0; iy < ny; iy++)
00117     dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00118 for (iz = 0; iz < nz; iz++)
00119     dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00120
00121 /* Write coordinates... */
00122 NC_PUT_DOUBLE("time", &t0, 0);
00123 NC_PUT_DOUBLE("lev", dataZ, 0);
00124 NC_PUT_DOUBLE("lat", dataLat, 0);
00125 NC_PUT_DOUBLE("lon", dataLon, 0);
00126
00127 /* Create wind fields (Williamson et al., 1992)... */
00128 for (ix = 0; ix < nx; ix++)
00129     for (iy = 0; iy < ny; iy++)
00130         for (iz = 0; iz < nz; iz++) {
00131             idx = (iz * ny + iy) * nx + ix;
00132             dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00133                 * (cos(dataLat[iy] * M_PI / 180.0)
00134                     * cos(alpha * M_PI / 180.0)
00135                     + sin(dataLat[iy] * M_PI / 180.0)
00136                         * cos(dataLon[ix] * M_PI / 180.0)
00137                         * sin(alpha * M_PI / 180.0)));
00138             dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00139                 * sin(dataLon[ix] * M_PI / 180.0)
00140                     * sin(alpha * M_PI / 180.0));
00141             dataW[idx] = (float) DZ2DP(1e-3 * w0, dataZ[iz]);
00142         }
00143
00144 /* Write data... */
00145 NC_PUT_FLOAT("T", dataT, 0);
00146 NC_PUT_FLOAT("U", dataU, 0);
00147 NC_PUT_FLOAT("V", dataV, 0);
00148 NC_PUT_FLOAT("W", dataW, 0);
00149
00150 /* Close file... */
00151 NC(nc_close(ncid));
00152
00153 /* Free... */
00154 free(dataT);
00155 free(dataU);
00156 free(dataV);
00157 free(dataW);
00158

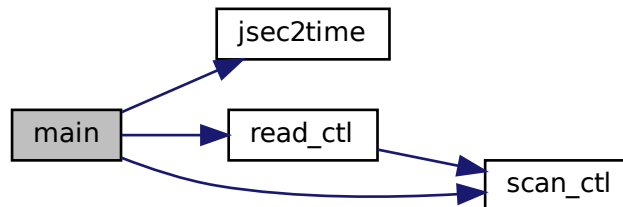
```

```

00159     return EXIT_SUCCESS;
00160 }

```

Here is the call graph for this function:



## 5.54 wind.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
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     static char filename[LEN];
00038
00039     static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00040         u0, u1, w0, alpha;
00041
00042     static float *dataT, *dataU, *dataV, *dataW;
00043
00044     static int ncid, varid, dims[4], idx, ix, iy, iz, nx, ny, nz,
00045         year, mon, day, hour, min, sec;
00046
00047     static size_t start[4], count[4];
00048
00049     /* Allocate... */
00050     ALLOC(dataT, float,
00051         EP * EY * EX);
00052     ALLOC(dataU, float,
00053         EP * EY * EX);
00054     ALLOC(dataV, float,
00055         EP * EY * EX);
00056     ALLOC(dataW, float,
00057         EP * EY * EX);
00058
00059     /* Check arguments... */

```

```

00060     if (argc < 3)
00061         ERRMSG("Give parameters: <ctl> <metbase>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065     t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00066     nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00067     ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00068     nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00069     z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00070     z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00071     u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00072     u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00073     w0 = scan_ctl(argv[1], argc, argv, "WIND_W0", -1, "0", NULL);
00074     alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00075
00076     /* Check dimensions... */
00077     if (nx < 1 || nx > EX)
00078         ERRMSG("Set 1 <= NX <= MAX!");
00079     if (ny < 1 || ny > EY)
00080         ERRMSG("Set 1 <= NY <= MAX!");
00081     if (nz < 1 || nz > EZ)
00082         ERRMSG("Set 1 <= NZ <= MAX!");
00083
00084     /* Get time... */
00085     jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00086     t0 = year * 10000. + mon * 100. + day + hour / 24.;
00087
00088     /* Set filename... */
00089     sprintf(filename, "%s_d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00090
00091     /* Create netCDF file... */
00092     NC(nc_create(filename, NC_CLOBBER, &ncid));
00093
00094     /* Create dimensions... */
00095     NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00096     NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00097     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00098     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00099
00100     /* Create variables... */
00101     NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time", "day as %Y%m%d.%f");
00102     NC_DEF_VAR("lev", NC_DOUBLE, 1, &dims[1], "air_pressure", "Pa");
00103     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[2], "latitude", "degrees_north");
00104     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[3], "longitude", "degrees_east");
00105     NC_DEF_VAR("T", NC_FLOAT, 4, &dims[0], "Temperature", "K");
00106     NC_DEF_VAR("U", NC_FLOAT, 4, &dims[0], "zonal wind", "m s**-1");
00107     NC_DEF_VAR("V", NC_FLOAT, 4, &dims[0], "meridional wind", "m s**-1");
00108     NC_DEF_VAR("W", NC_FLOAT, 4, &dims[0], "vertical velocity", "Pa s**-1");
00109
00110     /* End definition... */
00111     NC(nc_enddef(ncid));
00112
00113     /* Set coordinates... */
00114     for (ix = 0; ix < nx; ix++)
00115         dataLon[ix] = 360.0 / nx * (double) ix;
00116     for (iy = 0; iy < ny; iy++)
00117         dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00118     for (iz = 0; iz < nz; iz++)
00119         dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00120
00121     /* Write coordinates... */
00122     NC_PUT_DOUBLE("time", &t0, 0);
00123     NC_PUT_DOUBLE("lev", dataZ, 0);
00124     NC_PUT_DOUBLE("lat", dataLat, 0);
00125     NC_PUT_DOUBLE("lon", dataLon, 0);
00126
00127     /* Create wind fields (Williamson et al., 1992)... */
00128     for (ix = 0; ix < nx; ix++)
00129         for (iy = 0; iy < ny; iy++)
00130             for (iz = 0; iz < nz; iz++) {
00131                 idx = (iz * ny + iy) * nx + ix;
00132                 dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00133                     * (cos(dataLat[iy] * M_PI / 180.0)
00134                       * cos(alpha * M_PI / 180.0)
00135                       + sin(dataLat[iy] * M_PI / 180.0)
00136                       * cos(dataLon[ix] * M_PI / 180.0)
00137                       * sin(alpha * M_PI / 180.0)));
00138                 dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00139                     * sin(dataLon[ix] * M_PI / 180.0)
00140                     * sin(alpha * M_PI / 180.0));
00141                 dataW[idx] = (float) DZ2DP(1e-3 * w0, dataZ[iz]);
00142             }
00143
00144     /* Write data... */
00145     NC_PUT_FLOAT("T", dataT, 0);
00146     NC_PUT_FLOAT("U", dataU, 0);

```



```
00147     NC_PUT_FLOAT("V", dataV, 0);
00148     NC_PUT_FLOAT("W", dataW, 0);
00149
00150     /* Close file... */
00151     NC(nc_close(ncid));
00152
00153     /* Free... */
00154     free(dataT);
00155     free(dataU);
00156     free(dataV);
00157     free(dataW);
00158
00159     return EXIT_SUCCESS;
00160 }
```



## Index

advect  
    ctl\_t, 38  
ALLOC  
    libtrac.h, 292  
ARRAY\_2D  
    libtrac.h, 292  
ARRAY\_3D  
    libtrac.h, 292  
atm\_basename  
    ctl\_t, 46  
atm\_conv.c, 64, 66  
    main, 65  
atm\_dist.c, 67, 72  
    main, 67  
atm\_dt\_out  
    ctl\_t, 47  
atm\_filter  
    ctl\_t, 47  
atm\_gpfile  
    ctl\_t, 46  
atm\_init.c, 77, 79  
    main, 77  
atm\_select.c, 80, 83  
    main, 81  
atm\_split.c, 85, 87  
    main, 85  
atm\_stat.c, 89, 93  
    main, 90  
atm\_stride  
    ctl\_t, 47  
atm\_t, 3  
    lat, 4  
    lon, 4  
    np, 4  
    p, 4  
    q, 5  
    time, 4  
    zeta, 4  
atm\_type  
    ctl\_t, 47  
  
balloon  
    ctl\_t, 38  
bound\_dps  
    ctl\_t, 42  
bound\_lat0  
    ctl\_t, 42  
bound\_lat1  
    ctl\_t, 42  
bound\_mass  
    ctl\_t, 41  
bound\_mass\_trend  
    ctl\_t, 41  
bound\_p0  
    ctl\_t, 42  
  
bound\_p1  
    ctl\_t, 42  
bound\_vmr  
    ctl\_t, 41  
bound\_vmr\_trend  
    ctl\_t, 41  
  
cache\_t, 5  
    iso\_n, 6  
    iso\_ps, 6  
    iso\_ts, 6  
    iso\_var, 5  
    uvwp, 6  
cape  
    met\_t, 61  
cart2geo  
    libtrac.c, 104  
    libtrac.h, 308  
check\_finite  
    libtrac.h, 308  
chunkszhint  
    ctl\_t, 21  
cin  
    met\_t, 62  
cl  
    met\_t, 61  
clams\_met\_data  
    ctl\_t, 21  
clim\_h2o2  
    libtrac.c, 112  
    libtrac.h, 316  
clim\_h2o2\_filename  
    ctl\_t, 43  
clim\_h2o2\_init  
    libtrac.c, 113  
    libtrac.h, 317  
clim\_hno3  
    libtrac.c, 104  
    libtrac.h, 308  
clim\_hno3\_init  
    libtrac.c, 105  
    libtrac.h, 309  
clim\_oh  
    libtrac.c, 109  
    libtrac.h, 312  
clim\_oh\_diurnal  
    libtrac.c, 110  
    libtrac.h, 313  
clim\_oh\_filename  
    ctl\_t, 43  
clim\_oh\_init  
    libtrac.c, 110  
    libtrac.h, 314  
clim\_oh\_init\_help  
    libtrac.c, 111

- libtrac.h, 315
- clim\_t, 6
  - h2o2, 12
  - h2o2\_lat, 11
  - h2o2\_nlat, 11
  - h2o2\_np, 11
  - h2o2\_ntime, 11
  - h2o2\_p, 12
  - h2o2\_time, 11
  - hno3, 9
  - hno3\_lat, 9
  - hno3\_nlat, 9
  - hno3\_np, 9
  - hno3\_ntime, 8
  - hno3\_p, 9
  - hno3\_time, 9
  - oh, 11
  - oh\_lat, 10
  - oh\_nlat, 10
  - oh\_np, 10
  - oh\_ntime, 10
  - oh\_p, 10
  - oh\_time, 10
  - tropo, 8
  - tropo\_lat, 8
  - tropo\_nlat, 8
  - tropo\_ntime, 8
  - tropo\_time, 8
- clim\_tropo
  - libtrac.c, 114
  - libtrac.h, 318
- clim\_tropo\_init
  - libtrac.c, 115
  - libtrac.h, 319
- compress\_pack
  - libtrac.c, 117
  - libtrac.h, 321
- conv\_cape
  - ctl\_t, 40
- conv\_cin
  - ctl\_t, 40
- conv\_dt
  - ctl\_t, 40
- conv\_mix
  - ctl\_t, 40
- conv\_mix\_bot
  - ctl\_t, 41
- conv\_mix\_top
  - ctl\_t, 41
- conv\_wcape
  - ctl\_t, 40
- conv\_wmax
  - ctl\_t, 40
- CP
  - libtrac.h, 292
- CPD
  - libtrac.h, 287
- csi\_basename
  - ctl\_t, 47
- csi\_dt\_out
  - ctl\_t, 47
- csi\_lat0
  - ctl\_t, 49
- csi\_lat1
  - ctl\_t, 49
- csi\_lon0
  - ctl\_t, 49
- csi\_lon1
  - ctl\_t, 49
- csi\_modmin
  - ctl\_t, 48
- csi\_nx
  - ctl\_t, 49
- csi\_ny
  - ctl\_t, 49
- csi\_nz
  - ctl\_t, 48
- csi\_obsfile
  - ctl\_t, 48
- csi\_obsmin
  - ctl\_t, 48
- csi\_z0
  - ctl\_t, 48
- csi\_z1
  - ctl\_t, 48
- CT
  - libtrac.h, 292
- ctl\_t, 12
  - advect, 38
  - atm\_basename, 46
  - atm\_dt\_out, 47
  - atm\_filter, 47
  - atm\_gpfile, 46
  - atm\_stride, 47
  - atm\_type, 47
  - balloon, 38
  - bound\_dps, 42
  - bound\_lat0, 42
  - bound\_lat1, 42
  - bound\_mass, 41
  - bound\_mass\_trend, 41
  - bound\_p0, 42
  - bound\_p1, 42
  - bound\_vmr, 41
  - bound\_vmr\_trend, 41
  - chunkszhint, 21
  - clams\_met\_data, 21
  - clim\_h2o2\_filename, 43
  - clim\_oh\_filename, 43
  - conv\_cape, 40
  - conv\_cin, 40
  - conv\_dt, 40
  - conv\_mix, 40
  - conv\_mix\_bot, 41
  - conv\_mix\_top, 41
  - conv\_wcape, 40

conv\_wmax, [40](#)  
csi\_basename, [47](#)  
csi\_dt\_out, [47](#)  
csi\_lat0, [49](#)  
csi\_lat1, [49](#)  
csi\_lon0, [49](#)  
csi\_lon1, [49](#)  
csi\_modmin, [48](#)  
csi\_nx, [49](#)  
csi\_ny, [49](#)  
csi\_nz, [48](#)  
csi\_obsfile, [48](#)  
csi\_obsmin, [48](#)  
csi\_z0, [48](#)  
csi\_z1, [48](#)  
direction, [33](#)  
dry\_depo, [44](#)  
dt\_met, [33](#)  
dt\_mod, [33](#)  
ens\_basename, [50](#)  
ens\_dt\_out, [50](#)  
grid\_basename, [50](#)  
grid\_dt\_out, [50](#)  
grid\_gpfile, [50](#)  
grid\_lat0, [52](#)  
grid\_lat1, [52](#)  
grid\_lon0, [51](#)  
grid\_lon1, [51](#)  
grid\_nx, [51](#)  
grid\_ny, [52](#)  
grid\_nz, [51](#)  
grid\_sparse, [50](#)  
grid\_type, [52](#)  
grid\_z0, [51](#)  
grid\_z1, [51](#)  
h2o2\_chem\_cc, [44](#)  
h2o2\_chem\_reaction, [44](#)  
isosurf, [38](#)  
met\_cache, [38](#)  
met\_cloud, [37](#)  
met\_cloud\_min, [37](#)  
met\_detrend, [35](#)  
met\_dp, [34](#)  
met\_dt\_out, [37](#)  
met\_dx, [34](#)  
met\_dy, [34](#)  
met\_geopot\_sx, [35](#)  
met\_geopot\_sy, [36](#)  
met\_nc\_scale, [34](#)  
met\_np, [35](#)  
met\_p, [35](#)  
met\_sp, [35](#)  
met\_sx, [34](#)  
met\_sy, [35](#)  
met\_tropo, [36](#)  
met\_tropo\_lapse, [36](#)  
met\_tropo\_lapse\_sep, [36](#)  
met\_tropo\_nlev, [36](#)  
met\_tropo\_nlev\_sep, [36](#)  
met\_tropo\_pv, [37](#)  
met\_tropo\_spline, [37](#)  
met\_tropo\_theta, [37](#)  
met\_type, [34](#)  
metbase, [33](#)  
molmass, [43](#)  
nq, [22](#)  
oh\_chem, [44](#)  
oh\_chem\_beta, [44](#)  
oh\_chem\_reaction, [43](#)  
prof\_basename, [52](#)  
prof\_lat0, [54](#)  
prof\_lat1, [54](#)  
prof\_lon0, [53](#)  
prof\_lon1, [53](#)  
prof\_nx, [53](#)  
prof\_ny, [54](#)  
prof\_nz, [53](#)  
prof\_obsfile, [52](#)  
prof\_z0, [53](#)  
prof\_z1, [53](#)  
psc\_h2o, [46](#)  
psc\_hno3, [46](#)  
qnt\_cape, [28](#)  
qnt\_cin, [28](#)  
qnt\_cl, [27](#)  
qnt\_ens, [23](#)  
qnt\_format, [22](#)  
qnt\_h2o, [26](#)  
qnt\_h2ot, [25](#)  
qnt\_hno3, [28](#)  
qnt\_idx, [22](#)  
qnt\_iwc, [27](#)  
qnt\_lapse, [31](#)  
qnt\_longname, [22](#)  
qnt\_lwc, [27](#)  
qnt\_m, [23](#)  
qnt\_mloss\_decay, [30](#)  
qnt\_mloss\_dry, [29](#)  
qnt\_mloss\_h2o2, [29](#)  
qnt\_mloss\_oh, [29](#)  
qnt\_mloss\_wet, [29](#)  
qnt\_name, [22](#)  
qnt\_o3, [27](#)  
qnt\_oh, [29](#)  
qnt\_p, [25](#)  
qnt\_pbl, [24](#)  
qnt\_pcb, [27](#)  
qnt\_pct, [27](#)  
qnt\_pel, [28](#)  
qnt\_plcl, [28](#)  
qnt\_plfc, [28](#)  
qnt\_ps, [24](#)  
qnt\_psat, [30](#)  
qnt\_psice, [30](#)  
qnt\_pt, [25](#)  
qnt\_pv, [32](#)

qnt\_pw, 30  
qnt\_rh, 30  
qnt\_rhice, 31  
qnt\_rho, 26  
qnt\_rhop, 23  
qnt\_rp, 23  
qnt\_sh, 30  
qnt\_stat, 23  
qnt\_t, 26  
qnt\_tdew, 32  
qnt\_theta, 31  
qnt\_tice, 32  
qnt\_tnat, 32  
qnt\_ts, 24  
qnt\_tsts, 32  
qnt\_tt, 25  
qnt\_tvirt, 31  
qnt\_u, 26  
qnt\_unit, 22  
qnt\_us, 24  
qnt\_v, 26  
qnt\_vh, 31  
qnt\_vmr, 23  
qnt\_vmrimpl, 29  
qnt\_vs, 24  
qnt\_vz, 32  
qnt\_w, 26  
qnt\_z, 25  
qnt\_zeta, 31  
qnt\_zs, 24  
qnt\_zt, 25  
read\_mode, 21  
reflect, 38  
sample\_basename, 54  
sample\_dx, 54  
sample\_dz, 55  
sample\_obsfile, 54  
sort\_dt, 38  
species, 42  
stat\_basename, 55  
stat\_lat, 55  
stat\_lon, 55  
stat\_r, 55  
stat\_t0, 55  
stat\_t1, 56  
t\_start, 33  
t\_stop, 33  
tdec\_strat, 43  
tdec\_trop, 43  
turb\_dx\_strat, 39  
turb\_dx\_trop, 39  
turb\_dz\_strat, 39  
turb\_dz\_trop, 39  
turb\_mesox, 39  
turb\_mesoz, 39  
vert\_coord\_ap, 21  
vert\_coord\_met, 21  
vert\_vel, 21  
wet\_depo\_bc\_a, 45  
wet\_depo\_bc\_b, 45  
wet\_depo\_bc\_h, 45  
wet\_depo\_bc\_ret\_ratio, 46  
wet\_depo\_ic\_a, 45  
wet\_depo\_ic\_b, 45  
wet\_depo\_ic\_h, 45  
wet\_depo\_ic\_ret\_ratio, 46  
wet\_depo\_pre, 44  
CY  
    libtrac.h, 291  
day2doy  
    libtrac.c, 118  
    libtrac.h, 322  
day2doy.c, 96, 97  
    main, 96  
DEG2DX  
    libtrac.h, 293  
DEG2DY  
    libtrac.h, 293  
direction  
    ctl\_t, 33  
DIST  
    libtrac.h, 294  
DIST2  
    libtrac.h, 294  
DLAPSE  
    met\_lapse.c, 430  
DOTP  
    libtrac.h, 294  
doy2day  
    libtrac.c, 119  
    libtrac.h, 323  
doy2day.c, 97, 98  
    main, 97  
DP2DZ  
    libtrac.h, 293  
dry\_depo  
    ctl\_t, 44  
dt\_met  
    ctl\_t, 33  
dt\_mod  
    ctl\_t, 33  
DX2DEG  
    libtrac.h, 293  
DY2DEG  
    libtrac.h, 293  
DZ2DP  
    libtrac.h, 294  
ens\_basename  
    ctl\_t, 50  
ens\_dt\_out  
    ctl\_t, 50  
EP  
    libtrac.h, 290  
EPS  
    libtrac.h, 288

ERRMSG  
    libtrac.h, 306

EX  
    libtrac.h, 291

EY  
    libtrac.h, 291

fft\_help  
    met\_spec.c, 460

FMOD  
    libtrac.h, 294

FREAD  
    libtrac.h, 295

FWRITE  
    libtrac.h, 295

G0  
    libtrac.h, 288

geo2cart  
    libtrac.c, 119  
    libtrac.h, 323

get\_met  
    libtrac.c, 119  
    libtrac.h, 323

get\_met\_help  
    libtrac.c, 122  
    libtrac.h, 326

get\_met\_replace  
    libtrac.c, 123  
    libtrac.h, 327

get\_tropo  
    tropo.c, 538

grid\_basename  
    ctl\_t, 50

grid\_dt\_out  
    ctl\_t, 50

grid\_gpfile  
    ctl\_t, 50

grid\_lat0  
    ctl\_t, 52

grid\_lat1  
    ctl\_t, 52

grid\_lon0  
    ctl\_t, 51

grid\_lon1  
    ctl\_t, 51

grid\_nx  
    ctl\_t, 51

grid\_ny  
    ctl\_t, 52

grid\_nz  
    ctl\_t, 51

grid\_sparse  
    ctl\_t, 50

grid\_type  
    ctl\_t, 52

grid\_z0  
    ctl\_t, 51

grid\_z1  
    ctl\_t, 51

H0  
    libtrac.h, 288

h2o  
    met\_t, 63

h2o2  
    clim\_t, 12

h2o2\_chem\_cc  
    ctl\_t, 44

h2o2\_chem\_reaction  
    ctl\_t, 44

h2o2\_lat  
    clim\_t, 11

h2o2\_nlat  
    clim\_t, 11

h2o2\_np  
    clim\_t, 11

h2o2\_ntime  
    clim\_t, 11

h2o2\_p  
    clim\_t, 12

h2o2\_time  
    clim\_t, 11

h2ot  
    met\_t, 60

hno3  
    clim\_t, 9

hno3\_lat  
    clim\_t, 9

hno3\_nlat  
    clim\_t, 9

hno3\_np  
    clim\_t, 9

hno3\_ntime  
    clim\_t, 8

hno3\_p  
    clim\_t, 9

hno3\_time  
    clim\_t, 9

IDXMAX  
    met\_lapse.c, 430

INTPOL\_2D  
    libtrac.h, 295

INTPOL\_3D  
    libtrac.h, 296

INTPOL\_INIT  
    libtrac.h, 295

intpol\_met\_space\_2d  
    libtrac.c, 125  
    libtrac.h, 329

intpol\_met\_space\_3d  
    libtrac.c, 124  
    libtrac.h, 328

intpol\_met\_time\_2d  
    libtrac.c, 127  
    libtrac.h, 331

intpol\_met\_time\_3d

- libtrac.c, 126
- libtrac.h, 330
- INTPOL\_SPACE\_ALL
  - libtrac.h, 296
- INTPOL\_TIME\_ALL
  - libtrac.h, 296
- intpol\_tropo\_3d
  - tropo\_sample.c, 546
- iso\_n
  - cache\_t, 6
- iso\_ps
  - cache\_t, 6
- iso\_ts
  - cache\_t, 6
- iso\_var
  - cache\_t, 5
- isosurf
  - ctl\_t, 38
- iwc
  - met\_t, 63
- jsec2time
  - libtrac.c, 127
  - libtrac.h, 331
- jsec2time.c, 99, 100
  - main, 99
- KB
  - libtrac.h, 288
- LAPSE
  - libtrac.h, 297
- lapse\_rate
  - libtrac.c, 128
  - libtrac.h, 332
- LAPSEMIN
  - met\_lapse.c, 430
- lat
  - atm\_t, 4
  - met\_t, 58
- LEN
  - libtrac.h, 290
- libtrac.c, 100, 202
  - cart2geo, 104
  - clim\_h2o2, 112
  - clim\_h2o2\_init, 113
  - clim\_hno3, 104
  - clim\_hno3\_init, 105
  - clim\_oh, 109
  - clim\_oh\_diurnal, 110
  - clim\_oh\_init, 110
  - clim\_oh\_init\_help, 111
  - clim\_tropo, 114
  - clim\_tropo\_init, 115
  - compress\_pack, 117
  - day2doy, 118
  - doy2day, 119
  - geo2cart, 119
  - get\_met, 119
  - get\_met\_help, 122
  - get\_met\_replace, 123
  - intpol\_met\_space\_2d, 125
  - intpol\_met\_space\_3d, 124
  - intpol\_met\_time\_2d, 127
  - intpol\_met\_time\_3d, 126
  - jsec2time, 127
  - lapse\_rate, 128
  - locate\_irr, 128
  - locate\_reg, 129
  - nat\_temperature, 129
  - quicksort, 130
  - quicksort\_partition, 130
  - read\_atm, 131
  - read\_atm\_asc, 132
  - read\_atm\_bin, 133
  - read\_atm\_clams, 133
  - read\_atm\_nc, 134
  - read\_clim, 135
  - read\_ctl, 135
  - read\_met, 142
  - read\_met\_bin\_2d, 146
  - read\_met\_bin\_3d, 147
  - read\_met\_cape, 148
  - read\_met\_cloud, 149
  - read\_met\_detrend, 150
  - read\_met\_extrapolate, 152
  - read\_met\_geopot, 152
  - read\_met\_grid, 154
  - read\_met\_levels, 156
  - read\_met\_ml2pl, 158
  - read\_met\_nc\_2d, 159
  - read\_met\_nc\_3d, 160
  - read\_met\_pbl, 162
  - read\_met\_periodic, 163
  - read\_met\_pv, 163
  - read\_met\_sample, 165
  - read\_met\_surface, 166
  - read\_met\_tropo, 168
  - read\_obs, 170
  - scan\_ctl, 171
  - sedi, 172
  - spline, 172
  - stddev, 173
  - sza, 174
  - time2jsec, 174
  - timer, 175
  - tropo\_weight, 176
  - write\_atm, 177
  - write\_atm\_asc, 178
  - write\_atm\_bin, 179
  - write\_atm\_clams, 180
  - write\_atm\_nc, 182
  - write\_csi, 183
  - write\_ens, 186
  - write\_grid, 187
  - write\_grid\_asc, 190
  - write\_grid\_nc, 191



- write\_met, [192](#)
- write\_met\_bin\_2d, [194](#)
- write\_met\_bin\_3d, [194](#)
- write\_prof, [195](#)
- write\_sample, [198](#)
- write\_station, [200](#)
- libtrac.h, [279](#), [406](#)
  - ALLOC, [292](#)
  - ARRAY\_2D, [292](#)
  - ARRAY\_3D, [292](#)
  - cart2geo, [308](#)
  - check\_finite, [308](#)
  - clim\_h2o2, [316](#)
  - clim\_h2o2\_init, [317](#)
  - clim\_hno3, [308](#)
  - clim\_hno3\_init, [309](#)
  - clim\_oh, [312](#)
  - clim\_oh\_diurnal, [313](#)
  - clim\_oh\_init, [314](#)
  - clim\_oh\_init\_help, [315](#)
  - clim\_tropo, [318](#)
  - clim\_tropo\_init, [319](#)
  - compress\_pack, [321](#)
  - CP, [292](#)
  - CPD, [287](#)
  - CT, [292](#)
  - CY, [291](#)
  - day2doy, [322](#)
  - DEG2DX, [293](#)
  - DEG2DY, [293](#)
  - DIST, [294](#)
  - DIST2, [294](#)
  - DOTP, [294](#)
  - doy2day, [323](#)
  - DP2DZ, [293](#)
  - DX2DEG, [293](#)
  - DY2DEG, [293](#)
  - DZ2DP, [294](#)
  - EP, [290](#)
  - EPS, [288](#)
  - ERRMSG, [306](#)
  - EX, [291](#)
  - EY, [291](#)
  - FMOD, [294](#)
  - FREAD, [295](#)
  - FWRITE, [295](#)
  - G0, [288](#)
  - geo2cart, [323](#)
  - get\_met, [323](#)
  - get\_met\_help, [326](#)
  - get\_met\_replace, [327](#)
  - H0, [288](#)
  - INTPOL\_2D, [295](#)
  - INTPOL\_3D, [296](#)
  - INTPOL\_INIT, [295](#)
  - intpol\_met\_space\_2d, [329](#)
  - intpol\_met\_space\_3d, [328](#)
  - intpol\_met\_time\_2d, [331](#)
  - intpol\_met\_time\_3d, [330](#)
  - INTPOL\_SPACE\_ALL, [296](#)
  - INTPOL\_TIME\_ALL, [296](#)
  - jsec2time, [331](#)
  - KB, [288](#)
  - LAPSE, [297](#)
  - lapse\_rate, [332](#)
  - LEN, [290](#)
  - LIN, [297](#)
  - locate\_irr, [332](#)
  - locate\_reg, [333](#)
  - LOG, [305](#)
  - LOGLEV, [305](#)
  - LV, [288](#)
  - MA, [288](#)
  - MH2O, [289](#)
  - MO3, [289](#)
  - nat\_temperature, [333](#)
  - NC, [297](#)
  - NC\_DEF\_VAR, [298](#)
  - NC\_GET\_DOUBLE, [298](#)
  - NC\_INQ\_DIM, [298](#)
  - NC\_PUT\_ATT, [299](#)
  - NC\_PUT\_DOUBLE, [299](#)
  - NC\_PUT\_FLOAT, [300](#)
  - NC\_PUT\_INT, [299](#)
  - NCSI, [290](#)
  - NENS, [291](#)
  - NN, [300](#)
  - NOBS, [291](#)
  - NORM, [300](#)
  - NP, [290](#)
  - NQ, [290](#)
  - NTHREADS, [291](#)
  - NTIMER, [306](#)
  - NVTX\_POP, [307](#)
  - NVTX\_PUSH, [307](#)
  - P, [300](#)
  - P0, [289](#)
  - PRINT, [306](#)
  - PRINT\_TIMERS, [306](#)
  - PSAT, [301](#)
  - PSICE, [301](#)
  - PW, [301](#)
  - quicksort, [334](#)
  - quicksort\_partition, [334](#)
  - RA, [289](#)
  - RE, [289](#)
  - read\_atm, [335](#)
  - read\_atm\_asc, [336](#)
  - read\_atm\_bin, [337](#)
  - read\_atm\_clams, [337](#)
  - read\_atm\_nc, [338](#)
  - read\_clim, [339](#)
  - read\_ctl, [339](#)
  - read\_met, [346](#)
  - read\_met\_bin\_2d, [350](#)
  - read\_met\_bin\_3d, [351](#)

read\_met\_cape, 352  
 read\_met\_cloud, 353  
 read\_met\_detrrend, 354  
 read\_met\_extrapolate, 356  
 read\_met\_geopot, 356  
 read\_met\_grid, 358  
 read\_met\_levels, 360  
 read\_met\_ml2pl, 362  
 read\_met\_nc\_2d, 363  
 read\_met\_nc\_3d, 364  
 read\_met\_pbl, 366  
 read\_met\_periodic, 367  
 read\_met\_pv, 367  
 read\_met\_sample, 369  
 read\_met\_surface, 370  
 read\_met\_tropo, 372  
 read\_obs, 374  
 RH, 301  
 RHICE, 301  
 RHO, 302  
 RI, 289  
 scan\_ctl, 375  
 sedi, 376  
 SELECT\_TIMER, 306  
 SET\_ATM, 302  
 SET\_QNT, 302  
 SH, 302  
 spline, 376  
 SQR, 303  
 START\_TIMERS, 307  
 stddev, 377  
 STOP\_TIMERS, 307  
 SWAP, 303  
 sza, 378  
 T0, 290  
 TDEW, 303  
 THETA, 303  
 THETA\_VIRT, 304  
 thrustSortWrapper, 307  
 TICE, 303  
 time2jsec, 378  
 timer, 379  
 TOK, 304  
 tropo\_weight, 380  
 TVIRT, 304  
 WARN, 305  
 write\_atm, 381  
 write\_atm\_asc, 382  
 write\_atm\_bin, 383  
 write\_atm\_clams, 384  
 write\_atm\_nc, 386  
 write\_csi, 387  
 write\_ens, 390  
 write\_grid, 391  
 write\_grid\_asc, 394  
 write\_grid\_nc, 395  
 write\_met, 396  
 write\_met\_bin\_2d, 398  
 write\_met\_bin\_3d, 398  
 write\_prof, 399  
 write\_sample, 402  
 write\_station, 404  
 Z, 304  
 ZDIFF, 304  
 ZETA, 305  
 LIN  
   libtrac.h, 297  
 locate\_irr  
   libtrac.c, 128  
   libtrac.h, 332  
 locate\_reg  
   libtrac.c, 129  
   libtrac.h, 333  
 LOG  
   libtrac.h, 305  
 LOGLEV  
   libtrac.h, 305  
 lon  
   atm\_t, 4  
   met\_t, 58  
 LV  
   libtrac.h, 288  
 lwc  
   met\_t, 63  
 MA  
   libtrac.h, 288  
 main  
   atm\_conv.c, 65  
   atm\_dist.c, 67  
   atm\_init.c, 77  
   atm\_select.c, 81  
   atm\_split.c, 85  
   atm\_stat.c, 90  
   day2doy.c, 96  
   doy2day.c, 97  
   jsec2time.c, 99  
   met\_conv.c, 427  
   met\_lapse.c, 430  
   met\_map.c, 438  
   met\_prof.c, 446  
   met\_sample.c, 454  
   met\_spec.c, 461  
   met\_subgrid.c, 466  
   met\_zm.c, 473  
   sedi.c, 480  
   time2jsec.c, 482  
   tnat.c, 484  
   trac.c, 510  
   tropo.c, 539  
   tropo\_sample.c, 548  
   tropo\_zm.c, 556  
   wind.c, 561  
 met\_cache  
   ctl\_t, 38  
 met\_cloud  
   ctl\_t, 37

met\_cloud\_min  
  ctl\_t, 37  
met\_conv.c, 427, 428  
  main, 427  
met\_detrend  
  ctl\_t, 35  
met\_dp  
  ctl\_t, 34  
met\_dt\_out  
  ctl\_t, 37  
met\_dx  
  ctl\_t, 34  
met\_dy  
  ctl\_t, 34  
met\_geopot\_sx  
  ctl\_t, 35  
met\_geopot\_sy  
  ctl\_t, 36  
met\_lapse.c, 429, 434  
  DLAPSE, 430  
  IDXMAX, 430  
  LAPSEMIN, 430  
  main, 430  
met\_map.c, 437, 442  
  main, 438  
  NX, 438  
  NY, 438  
met\_nc\_scale  
  ctl\_t, 34  
met\_np  
  ctl\_t, 35  
met\_p  
  ctl\_t, 35  
met\_prof.c, 446, 450  
  main, 446  
  NZ, 446  
met\_sample.c, 453, 457  
  main, 454  
met\_sp  
  ctl\_t, 35  
met\_spec.c, 460, 463  
  fft\_help, 460  
  main, 461  
  PMAX, 460  
met\_subgrid.c, 465, 469  
  main, 466  
met\_sx  
  ctl\_t, 34  
met\_sy  
  ctl\_t, 35  
met\_t, 56  
  cape, 61  
  cin, 62  
  cl, 61  
  h2o, 63  
  h2ot, 60  
  iwc, 63  
  lat, 58  
  lon, 58  
  lwc, 63  
  np, 58  
  nx, 58  
  ny, 58  
  o3, 63  
  p, 59  
  patp, 64  
  pbl, 60  
  pcb, 61  
  pct, 60  
  pel, 61  
  pl, 63  
  plcl, 61  
  plfc, 61  
  ps, 59  
  pt, 60  
  pv, 63  
  t, 62  
  time, 58  
  ts, 59  
  tt, 60  
  u, 62  
  us, 59  
  v, 62  
  vs, 59  
  w, 62  
  z, 62  
  zeta, 64  
  zeta\_dot, 64  
  zs, 59  
  zt, 60  
met\_tropo  
  ctl\_t, 36  
met\_tropo\_lapse  
  ctl\_t, 36  
met\_tropo\_lapse\_sep  
  ctl\_t, 36  
met\_tropo\_nlev  
  ctl\_t, 36  
met\_tropo\_nlev\_sep  
  ctl\_t, 36  
met\_tropo\_pv  
  ctl\_t, 37  
met\_tropo\_spline  
  ctl\_t, 37  
met\_tropo\_theta  
  ctl\_t, 37  
met\_type  
  ctl\_t, 34  
met\_zm.c, 472, 476  
  main, 473  
  NY, 472  
  NZ, 472  
metbase  
  ctl\_t, 33  
MH2O  
  libtrac.h, 289

MO3  
     libtrac.h, 289  
 module\_advect  
     trac.c, 486  
 module\_bound\_cond  
     trac.c, 488  
 module\_convection  
     trac.c, 488  
 module\_decay  
     trac.c, 490  
 module\_diffusion\_meso  
     trac.c, 491  
 module\_diffusion\_turb  
     trac.c, 493  
 module\_dry\_deposition  
     trac.c, 493  
 module\_h2o2\_chem  
     trac.c, 499  
 module\_isosurf  
     trac.c, 496  
 module\_isosurf\_init  
     trac.c, 495  
 module\_meteo  
     trac.c, 497  
 module\_oh\_chem  
     trac.c, 498  
 module\_position  
     trac.c, 501  
 module\_rng  
     trac.c, 502  
 module\_rng\_init  
     trac.c, 502  
 module\_sedi  
     trac.c, 503  
 module\_sort  
     trac.c, 504  
 module\_sort\_help  
     trac.c, 505  
 module\_timesteps  
     trac.c, 506  
 module\_timesteps\_init  
     trac.c, 506  
 module\_wet\_deposition  
     trac.c, 507  
 molmass  
     ctl\_t, 43  
  
 nat\_temperature  
     libtrac.c, 129  
     libtrac.h, 333  
 NC  
     libtrac.h, 297  
 NC\_DEF\_VAR  
     libtrac.h, 298  
 NC\_GET\_DOUBLE  
     libtrac.h, 298  
 NC\_INQ\_DIM  
     libtrac.h, 298  
 NC\_PUT\_ATT  
     libtrac.h, 299  
 NC\_PUT\_DOUBLE  
     libtrac.h, 299  
 NC\_PUT\_FLOAT  
     libtrac.h, 300  
 NC\_PUT\_INT  
     libtrac.h, 299  
 NCSI  
     libtrac.h, 290  
 NENS  
     libtrac.h, 291  
 NN  
     libtrac.h, 300  
 NOBS  
     libtrac.h, 291  
 NORM  
     libtrac.h, 300  
 NP  
     libtrac.h, 290  
 np  
     atm\_t, 4  
     met\_t, 58  
 NQ  
     libtrac.h, 290  
 nq  
     ctl\_t, 22  
 NT  
     tropo\_sample.c, 546  
     tropo\_zm.c, 555  
 NTHREADS  
     libtrac.h, 291  
 NTIMER  
     libtrac.h, 306  
 NVTX\_POP  
     libtrac.h, 307  
 NVTX\_PUSH  
     libtrac.h, 307  
 NX  
     met\_map.c, 438  
 nx  
     met\_t, 58  
 NY  
     met\_map.c, 438  
     met\_zm.c, 472  
 ny  
     met\_t, 58  
 NZ  
     met\_prof.c, 446  
     met\_zm.c, 472  
  
 o3  
     met\_t, 63  
 oh  
     clim\_t, 11  
 oh\_chem  
     ctl\_t, 44  
 oh\_chem\_beta  
     ctl\_t, 44  
 oh\_chem\_reaction

- ctl\_t, 43
- oh\_lat
  - clim\_t, 10
- oh\_nlat
  - clim\_t, 10
- oh\_np
  - clim\_t, 10
- oh\_ntime
  - clim\_t, 10
- oh\_p
  - clim\_t, 10
- oh\_time
  - clim\_t, 10
- P
  - libtrac.h, 300
- p
  - atm\_t, 4
  - met\_t, 59
- P0
  - libtrac.h, 289
- patp
  - met\_t, 64
- pbl
  - met\_t, 60
- pcb
  - met\_t, 61
- pct
  - met\_t, 60
- pel
  - met\_t, 61
- pl
  - met\_t, 63
- plcl
  - met\_t, 61
- plfc
  - met\_t, 61
- PMAX
  - met\_spec.c, 460
- PRINT
  - libtrac.h, 306
- PRINT\_TIMERS
  - libtrac.h, 306
- prof\_basename
  - ctl\_t, 52
- prof\_lat0
  - ctl\_t, 54
- prof\_lat1
  - ctl\_t, 54
- prof\_lon0
  - ctl\_t, 53
- prof\_lon1
  - ctl\_t, 53
- prof\_nx
  - ctl\_t, 53
- prof\_ny
  - ctl\_t, 54
- prof\_nz
  - ctl\_t, 53
- prof\_obsfile
  - ctl\_t, 52
- prof\_z0
  - ctl\_t, 53
- prof\_z1
  - ctl\_t, 53
- ps
  - met\_t, 59
- PSAT
  - libtrac.h, 301
- psc\_h2o
  - ctl\_t, 46
- psc\_hno3
  - ctl\_t, 46
- PSICE
  - libtrac.h, 301
- pt
  - met\_t, 60
- pv
  - met\_t, 63
- PW
  - libtrac.h, 301
- q
  - atm\_t, 5
- qnt\_cape
  - ctl\_t, 28
- qnt\_cin
  - ctl\_t, 28
- qnt\_cl
  - ctl\_t, 27
- qnt\_ens
  - ctl\_t, 23
- qnt\_format
  - ctl\_t, 22
- qnt\_h2o
  - ctl\_t, 26
- qnt\_h2ot
  - ctl\_t, 25
- qnt\_hno3
  - ctl\_t, 28
- qnt\_idx
  - ctl\_t, 22
- qnt\_iwc
  - ctl\_t, 27
- qnt\_lapse
  - ctl\_t, 31
- qnt\_longname
  - ctl\_t, 22
- qnt\_lwc
  - ctl\_t, 27
- qnt\_m
  - ctl\_t, 23
- qnt\_mloss\_decay
  - ctl\_t, 30
- qnt\_mloss\_dry
  - ctl\_t, 29
- qnt\_mloss\_h2o2
  - ctl\_t, 29

qnt\_mloss\_oh  
    ctl\_t, 29  
qnt\_mloss\_wet  
    ctl\_t, 29  
qnt\_name  
    ctl\_t, 22  
qnt\_o3  
    ctl\_t, 27  
qnt\_oh  
    ctl\_t, 29  
qnt\_p  
    ctl\_t, 25  
qnt\_pbl  
    ctl\_t, 24  
qnt\_pcb  
    ctl\_t, 27  
qnt\_pct  
    ctl\_t, 27  
qnt\_pel  
    ctl\_t, 28  
qnt\_plcl  
    ctl\_t, 28  
qnt\_plfc  
    ctl\_t, 28  
qnt\_ps  
    ctl\_t, 24  
qnt\_psat  
    ctl\_t, 30  
qnt\_psice  
    ctl\_t, 30  
qnt\_pt  
    ctl\_t, 25  
qnt\_pv  
    ctl\_t, 32  
qnt\_pw  
    ctl\_t, 30  
qnt\_rh  
    ctl\_t, 30  
qnt\_rhice  
    ctl\_t, 31  
qnt\_rho  
    ctl\_t, 26  
qnt\_rhop  
    ctl\_t, 23  
qnt\_rp  
    ctl\_t, 23  
qnt\_sh  
    ctl\_t, 30  
qnt\_stat  
    ctl\_t, 23  
qnt\_t  
    ctl\_t, 26  
qnt\_tdew  
    ctl\_t, 32  
qnt\_theta  
    ctl\_t, 31  
qnt\_tice  
    ctl\_t, 32  
qnt\_tnat  
    ctl\_t, 32  
qnt\_ts  
    ctl\_t, 24  
qnt\_tsts  
    ctl\_t, 32  
qnt\_tt  
    ctl\_t, 25  
qnt\_tvirt  
    ctl\_t, 31  
qnt\_u  
    ctl\_t, 26  
qnt\_unit  
    ctl\_t, 22  
qnt\_us  
    ctl\_t, 24  
qnt\_v  
    ctl\_t, 26  
qnt\_vh  
    ctl\_t, 31  
qnt\_vmr  
    ctl\_t, 23  
qnt\_vmrimpl  
    ctl\_t, 29  
qnt\_vs  
    ctl\_t, 24  
qnt\_vz  
    ctl\_t, 32  
qnt\_w  
    ctl\_t, 26  
qnt\_z  
    ctl\_t, 25  
qnt\_zeta  
    ctl\_t, 31  
qnt\_zs  
    ctl\_t, 24  
qnt\_zt  
    ctl\_t, 25  
quicksort  
    libtrac.c, 130  
    libtrac.h, 334  
quicksort\_partition  
    libtrac.c, 130  
    libtrac.h, 334  
RA  
    libtrac.h, 289  
RE  
    libtrac.h, 289  
read\_atm  
    libtrac.c, 131  
    libtrac.h, 335  
read\_atm\_asc  
    libtrac.c, 132  
    libtrac.h, 336  
read\_atm\_bin  
    libtrac.c, 133  
    libtrac.h, 337  
read\_atm\_clams

- libtrac.c, 133
- libtrac.h, 337
- read\_atm\_nc
  - libtrac.c, 134
  - libtrac.h, 338
- read\_clim
  - libtrac.c, 135
  - libtrac.h, 339
- read\_ctl
  - libtrac.c, 135
  - libtrac.h, 339
- read\_met
  - libtrac.c, 142
  - libtrac.h, 346
- read\_met\_bin\_2d
  - libtrac.c, 146
  - libtrac.h, 350
- read\_met\_bin\_3d
  - libtrac.c, 147
  - libtrac.h, 351
- read\_met\_cape
  - libtrac.c, 148
  - libtrac.h, 352
- read\_met\_cloud
  - libtrac.c, 149
  - libtrac.h, 353
- read\_met\_detrend
  - libtrac.c, 150
  - libtrac.h, 354
- read\_met\_extrapolate
  - libtrac.c, 152
  - libtrac.h, 356
- read\_met\_geopot
  - libtrac.c, 152
  - libtrac.h, 356
- read\_met\_grid
  - libtrac.c, 154
  - libtrac.h, 358
- read\_met\_levels
  - libtrac.c, 156
  - libtrac.h, 360
- read\_met\_ml2pl
  - libtrac.c, 158
  - libtrac.h, 362
- read\_met\_nc\_2d
  - libtrac.c, 159
  - libtrac.h, 363
- read\_met\_nc\_3d
  - libtrac.c, 160
  - libtrac.h, 364
- read\_met\_pbl
  - libtrac.c, 162
  - libtrac.h, 366
- read\_met\_periodic
  - libtrac.c, 163
  - libtrac.h, 367
- read\_met\_pv
  - libtrac.c, 163
- libtrac.h, 367
- read\_met\_sample
  - libtrac.c, 165
  - libtrac.h, 369
- read\_met\_surface
  - libtrac.c, 166
  - libtrac.h, 370
- read\_met\_tropo
  - libtrac.c, 168
  - libtrac.h, 372
- read\_mode
  - ctl\_t, 21
- read\_obs
  - libtrac.c, 170
  - libtrac.h, 374
- reflect
  - ctl\_t, 38
- RH
  - libtrac.h, 301
- RHICE
  - libtrac.h, 301
- RHO
  - libtrac.h, 302
- RI
  - libtrac.h, 289
- sample\_basename
  - ctl\_t, 54
- sample\_dx
  - ctl\_t, 54
- sample\_dz
  - ctl\_t, 55
- sample\_obsfile
  - ctl\_t, 54
- scan\_ctl
  - libtrac.c, 171
  - libtrac.h, 375
- sedi
  - libtrac.c, 172
  - libtrac.h, 376
- sedi.c, 480, 481
  - main, 480
- SELECT\_TIMER
  - libtrac.h, 306
- SET\_ATM
  - libtrac.h, 302
- SET\_QNT
  - libtrac.h, 302
- SH
  - libtrac.h, 302
- sort\_dt
  - ctl\_t, 38
- species
  - ctl\_t, 42
- spline
  - libtrac.c, 172
  - libtrac.h, 376
- SQR
  - libtrac.h, 303

- START\_TIMERS
  - libtrac.h, [307](#)
- stat\_basename
  - ctl\_t, [55](#)
- stat\_lat
  - ctl\_t, [55](#)
- stat\_lon
  - ctl\_t, [55](#)
- stat\_r
  - ctl\_t, [55](#)
- stat\_t0
  - ctl\_t, [55](#)
- stat\_t1
  - ctl\_t, [56](#)
- stddev
  - libtrac.c, [173](#)
  - libtrac.h, [377](#)
- STOP\_TIMERS
  - libtrac.h, [307](#)
- SWAP
  - libtrac.h, [303](#)
- sza
  - libtrac.c, [174](#)
  - libtrac.h, [378](#)
- t
  - met\_t, [62](#)
- T0
  - libtrac.h, [290](#)
- t\_start
  - ctl\_t, [33](#)
- t\_stop
  - ctl\_t, [33](#)
- tdec\_strat
  - ctl\_t, [43](#)
- tdec\_trop
  - ctl\_t, [43](#)
- TDEW
  - libtrac.h, [303](#)
- THETA
  - libtrac.h, [303](#)
- THETA\_VIRT
  - libtrac.h, [304](#)
- thrustSortWrapper
  - libtrac.h, [307](#)
- TICE
  - libtrac.h, [303](#)
- time
  - atm\_t, [4](#)
  - met\_t, [58](#)
- time2jsec
  - libtrac.c, [174](#)
  - libtrac.h, [378](#)
- time2jsec.c, [482](#), [483](#)
  - main, [482](#)
- timer
  - libtrac.c, [175](#)
  - libtrac.h, [379](#)
- tnat.c, [483](#), [485](#)
  - main, [484](#)
- TOK
  - libtrac.h, [304](#)
- trac.c, [485](#), [515](#)
  - main, [510](#)
  - module\_advect, [486](#)
  - module\_bound\_cond, [488](#)
  - module\_convection, [488](#)
  - module\_decay, [490](#)
  - module\_diffusion\_meso, [491](#)
  - module\_diffusion\_turb, [493](#)
  - module\_dry\_deposition, [493](#)
  - module\_h2o2\_chem, [499](#)
  - module\_isosurf, [496](#)
  - module\_isosurf\_init, [495](#)
  - module\_meteo, [497](#)
  - module\_oh\_chem, [498](#)
  - module\_position, [501](#)
  - module\_rng, [502](#)
  - module\_rng\_init, [502](#)
  - module\_sedi, [503](#)
  - module\_sort, [504](#)
  - module\_sort\_help, [505](#)
  - module\_timesteps, [506](#)
  - module\_timesteps\_init, [506](#)
  - module\_wet\_deposition, [507](#)
  - write\_output, [508](#)
- tropo
  - clim\_t, [8](#)
- tropo.c, [538](#), [542](#)
  - get\_tropo, [538](#)
  - main, [539](#)
- tropo\_lat
  - clim\_t, [8](#)
- tropo\_nlat
  - clim\_t, [8](#)
- tropo\_nptime
  - clim\_t, [8](#)
- tropo\_sample.c, [546](#), [551](#)
  - intpol\_tropo\_3d, [546](#)
  - main, [548](#)
  - NT, [546](#)
- tropo\_time
  - clim\_t, [8](#)
- tropo\_weight
  - libtrac.c, [176](#)
  - libtrac.h, [380](#)
- tropo\_zm.c, [555](#), [558](#)
  - main, [556](#)
  - NT, [555](#)
- ts
  - met\_t, [59](#)
- tt
  - met\_t, [60](#)
- turb\_dx\_strat
  - ctl\_t, [39](#)
- turb\_dx\_trop
  - ctl\_t, [39](#)



- turb\_dz\_strat
  - ctl\_t, 39
- turb\_dz\_trop
  - ctl\_t, 39
- turb\_mesox
  - ctl\_t, 39
- turb\_mesoz
  - ctl\_t, 39
- TVIRT
  - libtrac.h, 304
- u
  - met\_t, 62
- us
  - met\_t, 59
- uvwp
  - cache\_t, 6
- v
  - met\_t, 62
- vert\_coord\_ap
  - ctl\_t, 21
- vert\_coord\_met
  - ctl\_t, 21
- vert\_vel
  - ctl\_t, 21
- vs
  - met\_t, 59
- w
  - met\_t, 62
- WARN
  - libtrac.h, 305
- wet\_depo\_bc\_a
  - ctl\_t, 45
- wet\_depo\_bc\_b
  - ctl\_t, 45
- wet\_depo\_bc\_h
  - ctl\_t, 45
- wet\_depo\_bc\_ret\_ratio
  - ctl\_t, 46
- wet\_depo\_ic\_a
  - ctl\_t, 45
- wet\_depo\_ic\_b
  - ctl\_t, 45
- wet\_depo\_ic\_h
  - ctl\_t, 45
- wet\_depo\_ic\_ret\_ratio
  - ctl\_t, 46
- wet\_depo\_pre
  - ctl\_t, 44
- wind.c, 561, 563
  - main, 561
- write\_atm
  - libtrac.c, 177
  - libtrac.h, 381
- write\_atm\_asc
  - libtrac.c, 178
  - libtrac.h, 382
- write\_atm\_bin
  - libtrac.c, 179
  - libtrac.h, 383
- write\_atm\_clams
  - libtrac.c, 180
  - libtrac.h, 384
- write\_atm\_nc
  - libtrac.c, 182
  - libtrac.h, 386
- write\_csi
  - libtrac.c, 183
  - libtrac.h, 387
- write\_ens
  - libtrac.c, 186
  - libtrac.h, 390
- write\_grid
  - libtrac.c, 187
  - libtrac.h, 391
- write\_grid\_asc
  - libtrac.c, 190
  - libtrac.h, 394
- write\_grid\_nc
  - libtrac.c, 191
  - libtrac.h, 395
- write\_met
  - libtrac.c, 192
  - libtrac.h, 396
- write\_met\_bin\_2d
  - libtrac.c, 194
  - libtrac.h, 398
- write\_met\_bin\_3d
  - libtrac.c, 194
  - libtrac.h, 398
- write\_output
  - trac.c, 508
- write\_prof
  - libtrac.c, 195
  - libtrac.h, 399
- write\_sample
  - libtrac.c, 198
  - libtrac.h, 402
- write\_station
  - libtrac.c, 200
  - libtrac.h, 404
- Z
  - libtrac.h, 304
- z
  - met\_t, 62
- ZDIFF
  - libtrac.h, 304
- ZETA
  - libtrac.h, 305
- zeta
  - atm\_t, 4
  - met\_t, 64
- zeta\_dot
  - met\_t, 64
- zs

met\_t, [59](#)  
zt  
met\_t, [60](#)