

CORSIKA add-on package IACT/ATMO:

Version 1.60 (September 2019)

Generated by Doxygen 1.8.15

1 Module Index	1
1.1 Modules	1
2 Data Structure Index	2
2.1 Data Structures	2
3 File Index	3
3.1 File List	3
4 Module Documentation	5
4.1 The CORSIKA iact/atmo interface	5
4.1.1 Detailed Description	13
4.1.2 Macro Definition Documentation	13
4.1.3 Function Documentation	14
4.1.4 Variable Documentation	32
4.2 The fake_corsika program	36
4.2.1 Detailed Description	37
4.2.2 Function Documentation	37
4.3 The listio program	38
4.3.1 Detailed Description	38
4.3.2 Function Documentation	38
4.4 The read_iact program	39
4.4.1 Detailed Description	39
4.4.2 Function Documentation	39
4.5 The sim_skeleton program	40
4.5.1 Detailed Description	41
4.5.2 Macro Definition Documentation	41
4.5.3 Function Documentation	41
4.6 The testio program	44
4.6.1 Detailed Description	44
4.6.2 Function Documentation	44
5 Data Structure Documentation	49
5.1 _struct_IO_BUFFER Struct Reference	49
5.1.1 Detailed Description	50
5.1.2 Field Documentation	50
5.2 _struct_IO_ITEM_HEADER Struct Reference	52
5.2.1 Detailed Description	53
5.2.2 Field Documentation	53
5.3 atmospheric_profile Struct Reference	54
5.3.1 Detailed Description	55
5.3.2 Field Documentation	55
5.4 bunch Struct Reference	55
5.4.1 Detailed Description	56

5.5 camera_electronics Struct Reference	56
5.5.1 Field Documentation	56
5.6 compact_bunch Struct Reference	56
5.6.1 Detailed Description	57
5.7 cubic_params Struct Reference	57
5.8 detstruct Struct Reference	57
5.8.1 Detailed Description	58
5.9 ev_reg_chain Struct Reference	58
5.10 ev_reg_entry Struct Reference	59
5.11 gridstruct Struct Reference	59
5.12 incpath Struct Reference	60
5.12.1 Detailed Description	60
5.13 linked_string Struct Reference	61
5.13.1 Detailed Description	61
5.14 mc_options Struct Reference	61
5.14.1 Detailed Description	61
5.14.2 Field Documentation	61
5.15 mc_run Struct Reference	62
5.15.1 Field Documentation	63
5.16 photo_electron Struct Reference	65
5.16.1 Detailed Description	66
5.16.2 Field Documentation	66
5.17 photon_bunch Struct Reference	66
5.18 pm_camera Struct Reference	67
5.19 rpol_table Struct Reference	67
5.19.1 Detailed Description	69
5.19.2 Field Documentation	69
5.20 rpt_list Struct Reference	71
5.20.1 Detailed Description	71
5.21 shower_extra_parameters Struct Reference	71
5.21.1 Detailed Description	72
5.21.2 Field Documentation	72
5.22 simulated_shower_parameters Struct Reference	73
5.22.1 Field Documentation	74
5.23 telescope_array Struct Reference	75
5.23.1 Detailed Description	76
5.23.2 Field Documentation	76
5.24 telescope_optics Struct Reference	78
5.25 telpos_struct Struct Reference	78
5.26 test_struct Struct Reference	78
5.27 warn_specific_data Struct Reference	79
5.27.1 Detailed Description	79

5.27.2 Field Documentation	79
6 File Documentation	79
6.1 atmo.c File Reference	79
6.1.1 Detailed Description	83
6.2 atmo.h File Reference	84
6.2.1 Detailed Description	86
6.3 eventio_registry.c File Reference	86
6.3.1 Detailed Description	87
6.3.2 Function Documentation	87
6.4 eventio_registry.h File Reference	89
6.4.1 Detailed Description	90
6.4.2 Function Documentation	90
6.5 fake_corsika.c File Reference	91
6.5.1 Detailed Description	93
6.6 fileopen.c File Reference	93
6.6.1 Detailed Description	94
6.6.2 Function Documentation	95
6.6.3 Variable Documentation	99
6.7 fileopen.h File Reference	99
6.7.1 Detailed Description	100
6.7.2 Function Documentation	100
6.8 iact.c File Reference	103
6.8.1 Detailed Description	108
6.9 iact.h File Reference	109
6.9.1 Detailed Description	111
6.10 initial.h File Reference	111
6.10.1 Detailed Description	112
6.11 io_basic.h File Reference	113
6.11.1 Detailed Description	118
6.11.2 Macro Definition Documentation	119
6.11.3 Function Documentation	119
6.12 io_simtel.c File Reference	140
6.12.1 Detailed Description	143
6.12.2 Function Documentation	143
6.12.3 Variable Documentation	160
6.13 listino.c File Reference	161
6.13.1 Detailed Description	162
6.14 mc_atmprof.c File Reference	162
6.14.1 Function Documentation	163
6.15 mc_atmprof.h File Reference	164
6.15.1 Detailed Description	165

6.15.2 Function Documentation	165
6.16 mc_tel.h File Reference	166
6.16.1 Detailed Description	169
6.16.2 Function Documentation	169
6.17 read_iact.c File Reference	186
6.17.1 Detailed Description	186
6.18 rpolator.c File Reference	187
6.18.1 Detailed Description	189
6.18.2 Function Documentation	189
6.19 rpolator.h File Reference	200
6.19.1 Detailed Description	202
6.19.2 Function Documentation	202
6.20 sampling.c File Reference	212
6.20.1 Detailed Description	212
6.21 sampling.h File Reference	213
6.22 sim_skeleton.c File Reference	213
6.22.1 Detailed Description	215
6.23 straux.c File Reference	215
6.23.1 Detailed Description	216
6.23.2 Function Documentation	216
6.24 straux.h File Reference	217
6.24.1 Detailed Description	218
6.24.2 Function Documentation	218
6.25 testio.c File Reference	220
6.25.1 Detailed Description	222
6.26 warning.c File Reference	222
6.26.1 Detailed Description	223
6.26.2 Function Documentation	224
6.26.3 Variable Documentation	227
6.27 warning.h File Reference	227
6.27.1 Detailed Description	228
6.27.2 Function Documentation	229

1 Module Index

1.1 Modules

Here is a list of all modules:

The CORSIKA iact/atmo interface	5
The fake_corsika program	36

The listio program	38
The read_iact program	39
The sim_skeleton program	40
The testio program	44

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_struct_IO_BUFFER	
The <code>IO_BUFFER</code> structure contains all data needed the manage the stuff	49
_struct_IO_ITEM_HEADER	
An <code>IO_ITEM_HEADER</code> is to access header info for an I/O block and as a handle to the I/O buffer	52
atmospheric_profile	
Atmospheric profile as stored in <code>atmprof*.dat</code> files - the actually used columns only	54
bunch	
Photons collected in bunches of identical direction, position, time, and wavelength	55
camera_electronics	
Parameters of the electronics of a telescope	56
compact_bunch	
The <code>compact_bunch</code> struct is equivalent to the <code>bunch</code> struct except that we try to use less memory	56
cubic_params	
Cubic spline interpolation (natural cubic splines = scheme 3, clamped cubic splines = scheme 4)	57
detstruct	
A structure describing a detector and linking its photons bunches to it	57
ev_reg_chain	
Use a double-linked list for the registry	58
ev_reg_entry	59
gridstruct	59
incpath	
An element in a linked list of include paths	60
linked_string	
The <code>linked_string</code> is mainly used to keep CORSIKA input	61
mc_options	
Options of the simulation passed through to low-level functions	61
mc_run	
Basic parameters of the CORSIKA run	62

photo_electron	A photo-electron produced by a photon hitting a pixel	65
photon_bunch		66
pm_camera	Parameters of a telescope camera (pixels, ...)	67
rpol_table	Structure describing an interpolation table, interpolation scheme and selected options	67
rpt_list	Registered interpolation tables allow re-use of tables without having to load them again	71
shower_extra_parameters	Extra shower parameters of unspecified nature	71
simulated_shower_parameters	Basic parameters of a simulated shower	73
telescope_array	Description of telescope position, array offsets and shower parameters	75
telescope_optics	Parameters describing the telescope optics	78
telpos_struct		78
test_struct		78
warn_specific_data	A struct used to store thread-specific data	79

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

atmo.c	Use of tabulated atmospheric profiles and atmospheric refraction	79
atmo.h	Use of tabulated atmospheric profiles and atmospheric refraction	84
eventio_registry.c	Register and enquire about well-known I/O block types	86
eventio_registry.h	Register and enquire about well-known I/O block types	89
fake_corsika.c	Simple demonstration of how to write photon bunches without CORSIKA	91
fileopen.c	Allow searching of files in declared include paths (fopen replacement)	93

fileopen.h	Function prototypes for fileopen.c	99
iact.c	CORSIKA interface for Imaging Atmospheric Cherenkov Telescopes etc	103
iact.h	Function declarations for CORSIKA IACT interface	109
initial.h	Identification of the system and including some basic include file	111
io_basic.h	Basic header file for eventio data format	113
io_simtel.c	Write and read CORSIKA blocks and simulated Cherenkov photon bunches	140
listio.c	Main function for listing data consisting of eventio blocks	161
mc_atmprof.c	Interface to the atmospheric profile structure	162
mc_atmprof.h	A data structure shared between io_simtel.c and atmo.c - which is used by both sim_telarray and the CORSIKA IACT/atmo package	164
mc_tel.h	Definitions and structures for CORSIKA Cherenkov light interface	166
read_iact.c	A program reading simulated CORSIKA data written through the IACT interface and shows the contents as readable text	186
rpolator.c	Reading of configuration data tables and interpolation	187
rpolator.h	Memory structure and interfaces for rpolator interpolation code	200
sampling.c	Interface for importance sampled core offset distributions	212
sampling.h		213
sim_skeleton.c	A (non-functional) skeleton program for reading CORSIKA IACT data	213
straux.c	Check for abbreviations of strings and get words from strings	215
straux.h	Check for abbreviations of strings and get words from strings	217
testio.c	Test program for eventio data format	220
warning.c	Pass warning messages to the screen or a usr function as set up	222

[warning.h](#)

Pass warning messages to the screen or a usr function as set up

227

4 Module Documentation

4.1 The CORSIKA iact/atmo interface

Data Structures

- struct [detstruct](#)
A structure describing a detector and linking its photons bunches to it.
- struct [gridstruct](#)

Macros

- #define [EXTERNAL_STORAGE](#) 1
Enable external temporary bunch storage.
- #define **EXTRA_MEM** 0
- #define **EXTRA_MEM_1** EXTRA_MEM
- #define **EXTRA_MEM_10** EXTRA_MEM
- #define **EXTRA_MEM_11** EXTRA_MEM
- #define **EXTRA_MEM_12** EXTRA_MEM
- #define **EXTRA_MEM_2** EXTRA_MEM
- #define **EXTRA_MEM_3** EXTRA_MEM
- #define **EXTRA_MEM_4** EXTRA_MEM
- #define **EXTRA_MEM_5** EXTRA_MEM
- #define **EXTRA_MEM_6** EXTRA_MEM
- #define **EXTRA_MEM_7** EXTRA_MEM
- #define **EXTRA_MEM_8** EXTRA_MEM
- #define **EXTRA_MEM_9** EXTRA_MEM
- #define [GRID_SIZE](#) 1000
unit: cm
- #define **HAVE_EVENTIO_FUNCTIONS** 1
- #define **IACT_ATMO_VERSION** "1.60 (2019-07-24)"
- #define [INTERNAL_LIMIT](#) 100000
Start external storage after so many bunches.
- #define **M_PI** 3.14159265358979323846 /* pi */
- #define **max**(a, b) ((a)>(b)?(a):(b))
- #define [MAX_ARRAY_SIZE](#) 1000 /* Use the same limit as in CORSIKA itself. */
Maximum number of telescopes (or other detectors) per array.
- #define **MAX_BUNCHES** 5000000
- #define **MAX_CLASS** 1
- #define **MAX_IO_BUFFER** 200000000
- #define **min**(a, b) ((a)<(b)?(a):(b))
- #define [NBUNCH](#) 5000
Memory allocation step size for bunches.
- #define **PIPE_OUTPUT** 1
- #define **PRMPAR_SIZE** 17
- #define **square**(x) ((x)*(x))
- #define **UNKNOWN_LONG_DIST** 1

Typedefs

- typedef float `cors_real_t`
*Type for CORSIKA floating point numbers remaining REAL*4.*

Functions

- static int `compact_photon_hit` (struct `detstruct` *det, double x, double y, double cx, double cy, double sx, double sy, double photons, double ctime, double zem, double lambda)
Store a photon bunch for a given telescope in compact format.
- static void `cross_prod` (double *v1, double *v2, double *v3)
Cross (outer) product of two 3-D vectors v1, v2 into 3-D vector v3.
- static int `expand_env` (char *fname, size_t maxlen)
Expanding environment variables ourselves rather than passing that on a shell later, so that we can still check characters after expansion.
- void `extprim_setup` (const char *text)
Placeholder function for activating and setting up user-defined (external to CORSIKA) controlled over types, spectra, and angular distribution of primaries.
- void `extprm_` (`cors_dbl_t` *type, `cors_dbl_t` *eprim, double *thetap, double *phip)
Placeholder function for external shower-by-shower setting of primary type, energy, and direction.
- void `get_iact_stats` (long *sb, double *ab, double *ap)
- void `get_impact_offset` (`cors_real_t` evth[273], `cors_dbl_t` prmpar[PRMPAR_SIZE])
Approximate impact offset of primary due to geomagnetic field.
- static void `get_mass_charge` (int type, double *mass, double *charge)
- double `heigh_` (double *thickness)
The CORSIKA built-in function for the height as a function of overburden.
- static void `iact_param` (const char *text0)
Processing of IACT module specific parameters in Corsika input.
- double `iact_rndm` (int dummy)
- static int `in_detector` (struct `detstruct` *det, double x, double y, double sx, double sy)
Check if a photon bunch hits a particular telescope volume.
- static void `ioerrorcheck` ()
- static int `is_off` (char *word)
- static int `is_on` (char *word)
- static int `Nint_f` (double x)
Nearest integer function.
- static double `norm3` (double *v)
Norm of a 3-D vector.
- static void `norm_vec` (double *v)
Normalize a 3-D vector.
- static int `photon_hit` (struct `detstruct` *det, double x, double y, double cx, double cy, double sx, double sy, double photons, double ctime, double zem, double lambda)
Store a photon bunch for a given telescope in long format.
- double `refidx_` (double *height)
Index of refraction as a function of altitude [cm].
- double `rhof_` (double *height)
The CORSIKA built-in density lookup function.
- void `rmmard_` (double *, int *, int *)
- static double `rndm` (int dummy)
Random number interface using sequence 4 of CORSIKA.

- void `sample_offset` (const char *`sampling_fname`, double `core_range`, double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, double *xoff, double *yoff, double *sampling←
_area)
Get uniformly sampled or importance sampled offset of array with respect to core, in the plane perpendicular to the shower axis.
- static int `set_random_systems` (double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, int volflag)
Randomly scatter each array of detectors in given area.
- void `set_system_displacement` (double dx, double dy)
Report displaced core positions in event header.
- void `telasu_` (int *n, `cors_dbl_t` *dx, `cors_dbl_t` *dy)
Setup how many times each shower is used.
- void `telend_` (`cors_real_t` evte[273])
End of event.
- void `televt_` (`cors_real_t` evth[273], `cors_dbl_t` prmpar[PRMPAR_SIZE])
Start of new event.
- void `telfil_` (const char *name0)
Define the output file for photon bunches hitting the telescopes.
- void `telinf_` (int *itel, double *x, double *y, double *z, double *r, int *exists)
Return information about configured telescopes back to CORSIKA.
- void `telling_` (int *type, double *data, int *ndim, int *np, int *nthick, double *thickstep)
Write CORSIKA 'longitudinal' (vertical) distributions.
- void `tellni_` (const char *line, int *llength)
Keep a record of CORSIKA input lines.
- int `telout_` (`cors_dbl_t` *bsize, `cors_dbl_t` *wt, `cors_dbl_t` *px, `cors_dbl_t` *py, `cors_dbl_t` *pu, `cors_dbl_t` *pv, `cors_dbl_t` *ctime, `cors_dbl_t` *zem, `cors_dbl_t` *lambda)
Check if a photon bunch hits one or more simulated detector volumes.
- void `telrne_` (`cors_real_t` rune[273])
Write run end block to the output file.
- void `telrnh_` (`cors_real_t` runh[273])
Save aparameters from CORSIKA run header.
- void `telset_` (`cors_dbl_t` *x, `cors_dbl_t` *y, `cors_dbl_t` *z, `cors_dbl_t` *r)
Add another telescope to the system (array) of telescopes.
- void `telshw_` ()
Show what telescopes have actually been set up.
- void `telsmp_` (const char *name)

Variables

- static double `airlightspeed` = 29.9792458/1.0002256
[cm/ns] at H=2200 m
- static double `all_bunches`
- static double `all_bunches_run`
- static double `all_photons`
- static double `all_photons_run`
- static double `arr_cosang`
- static double `arr_sinang`
- static double `arrang`
- static double `atmo_top`
- static double `Bfield` [3]
Magnetic field vector in detector coordinate system (with Bz positive if upwards)

- static double **bxplane** [3]
- static double **byplane** [3]

Spanning vectors of shower plane such that projection of B field is in bxplane direction.

- static double **core_range**

The maximum core offset of array centres in circular distribution.

- static double **core_range1**

The maximum core offsets in x,y for rectangular distribution.

- static double **core_range2**
- int **cors_4dig_def** = 6900
- double **cors_ver_def** = 6.900
- static struct **linked_string corsika_inputs** = { corsika_inputs_head, NULL }
- static char **corsika_inputs_head** [80]
- int **corsika_version** = (CORSIKA_VERSION)

The CORSIKA version actually running.

- static int **count_print_evt** = 0
- static int **count_print_tel** = 0
- static int **det_in_class** [MAX_CLASS]
- static struct **detstruct ** detector**
- static double **dmax** = 0.

Max.

- static int **do_print**
- static double **energy**
- static int **event_number**
- static double **first_int**
- static struct **gridstruct * grid**
- static int **grid_elements**
- static int **grid_nx**
- static int **grid_ny**
- static double **grid_x_high**
- static double **grid_x_low**
- static double **grid_y_high**
- static double **grid_y_low**
- static int **impact_correction** = 1

Correct impact position if non-zero.

- static double **impact_offset** [2]

Offset of impact position of charged primaries.

- static **IO_BUFFER * iobuf**
- static double **lambda1**
- static double **lambda2**
- static long **max_bunches** = MAX_BUNCHES
- static int **max_internal_bunches** = INTERNAL_LIMIT

The largest number of photon bunches kept in main memory before attempting to flush them to temporary files on disk.

- static size_t **max_io_buffer** = MAX_IO_BUFFER

The largest block size in the output data, which must hold all photons bunches of one array.

- static int **max_print_evt** = 100
- static int **max_print_tel** = 10
- static int **narray**
- static int * **ndet**
- static int **nevents**
- static int **nrun**
- static int **nsys** = 1

Number of arrays.

- static int `ntel` = 0
Number of telescopes set up.
- static double `obs_dx`
- static double `obs_dy`
- static double `obs_height`
- static double `off_axis`
- static char * `output_fname` = NULL
The name of the output file for eventio format data.
- static char * `output_options` = NULL
- static double `phi_central`
- static double `phi_prim`
- static double `pprim` [3]
Momentum vector of primary particle.
- static int `primary`
- static double `raise_tel`
Non-zero if any telescope has negative z.
- static double `rmax` = 0.
Max.
- static double `rtel` [MAX_ARRAY_SIZE]
- static char * `sampling_fname`
The name of the file providing parameters for importance sampling.
- static int `skip_off2` = 1
- static int `skip_print` = 1
- static int `skip_print2` = 100
- static long `stored_bunches`
- static int `tel_individual` = 0
- static size_t `tel_split_threshold` = 10000000
- static int `televt_done`
- static double `theta_central`
The central value of the allowed ranges in theta and phi.
- static double `theta_prim`
- static double `toffset`
- static int `use_compact_format` = 1
- static double `ush`
- static double `ushc`
- static double `vsh`
- static double `vshc`
- static double * `weight`
- int `with_extprim` = 0
- static double `wsh`
- static double `wshc`
- static double `xdisplaced` = 0.
Extra reported core offset displacement (not used with CORSIKA)
- static double * `xoffset`
- static double `xtel` [MAX_ARRAY_SIZE]
Position and size definition of fiducial spheres.
- static double `ydisplaced` = 0.
- static double * `yoffset`
- static double `ytel` [MAX_ARRAY_SIZE]
- static double `ztel` [MAX_ARRAY_SIZE]
- int `atmosphere`

- The atmospheric profile number, 0 for built-in.*

 - static char * **atmprof_name** = NULL

File name for atmospheric profile table.
- static int **num_prof**

Number of levels in original table.
- static double **p_alt** [MAX_PROFILE]

Altitude levels in given table (converted to [cm], upward order).
- static double **p_alt_rev** [MAX_PROFILE]

Altitude levels in given table (converted to [cm], reversed order).
- static double **p_rho** [MAX_PROFILE]

Density at given levels [g/cm³] (upward height order)
- static double **p_rho_rev** [MAX_PROFILE]

Density at given levels [g/cm³], reversed order.
- static double **p_thick** [MAX_PROFILE]

Atmospheric thickness at given levels [g/cm²].
- static double **p_log_rho** [MAX_PROFILE]

Log of density at given levels (for better interpolation)
- static double **p_log_thick** [MAX_PROFILE]

Log of atmospheric thickness at given levels (upward height order)
- static double **p_log_thick_rev** [MAX_PROFILE]

Log of atmospheric thickness at given levels (reversed order)
- static double **p_log_n1** [MAX_PROFILE]

Log of (index of refraction minus 1.0) at given levels.
- static double **p_bend_ray_hori_a** [MAX_PROFILE]

Coefficient for horizontal displacement refraction effect (reversed order)
- static double **p_bend_ray_time_a** [MAX_PROFILE]

Coefficient for arrival time effect by refraction, density dependence (reversed order)
- static double **p_bend_ray_time0** [MAX_PROFILE]

Coefficient for arrival time effect by refraction, altitude dependence.
- static **CsplinePar** * **cs_alt_log_rho**

Cubic spline parameters for log(density) versus altitude interpolation.
- static **CsplinePar** * **cs_alt_log_thick**

Cubic spline parameters for log(thickness) versus altitude interpolation.
- static **CsplinePar** * **cs_alt_log_n1**

Cubic spline parameters for log(n-1) versus altitude interpolation.
- static **CsplinePar** * **cs_log_thick_alt**

Cubic spline parameters for altitude versus log(thickness) reverse interpolation.
- static **CsplinePar** * **cs_bend_rho_hori_a**

Cubic spline parameters for horizontal displacement refraction effect.
- static **CsplinePar** * **cs_bend_rho_time_a**

Cubic spline parameters for arrival time effect by refraction, density dependence.
- static **CsplinePar** * **cs_bend_alt_time0**

Cubic spline parameters for arrival time effect by refraction, altitude dependence.
- static double **top_of_atmosphere** = 112.83e5

Height of top of atmosphere [cm], = p_alt[num_prof-1].
- static double **top_of_atm_table**

The highest entry in the atmospheric profile table.
- static double **bottom_of_atmosphere** = 0.

Bottom is normally at sea level but table could differ, = p_alt[0].
- static double **bottom_log_thickness**

Thickness at bottom, depending on profile, = p_log_thick[0].

- static double `top_log_thickness`
Thickness at top is zero, thus using an extrapolation from the second last value.
- static double `top_layer_2nd_altitude`
Second highest altitude tabulated, using exponential density profile above that.
- static double `top_layer_rho0`
Sea-level altitude matching profile in top layer, without scaling for thickness.
- static double `top_layer_cfacs`
Scaling factor needed to match thickness at second last level.
- static double `top_layer_hscale`
Height scale of exponention density profile in top layer.
- static double `top_layer_hscale_rho0_cfacs`
 $top_layer_rho0 * top_layer_cfacs * top_layer_hscale$
- static double `top_layer_hscale_rho0_cfacs_inv`
1.
- static double `top_layer_exp_top`
 $exp(-top_of_atmosphere/top_layer_hscale)$
- static double * `fast_p_alt`
Equidistant steps in altitude.
- static double * `fast_p_log_rho`
Log(rho) at fast_p_alt steps.
- static double * `fast_p_log_thick`
Log(thick) at fast_p_alt steps.
- static double * `fast_p_log_n1`
Log(n-1) at fast_p_alt steps.
- static double `fast_h_fac`
Inverse of height difference per step.
- static double * `fast_p_thick`
Direct interpolation of thickness at fast_p_alt steps.
- static double * `fast_p_rho`
Direct interpolation of density at fast_p_alt steps.
- static double * `fast_p_n1`
Direct interpolation of n-1 at fast_p_alt steps.
- static double * `fast_p_eq_log_thick`
Equidistant steps in log(thick)
- static double * `fast_p_heigh_rev`
Altitude at given log(thick) steps.
- static double `fast_log_thick_fac`
Inverse of log(thickness) difference per step.
- static double * `fast_p_bend_ray_hori_a`
Coefficient for horizontal displacement refraction effect.
- static double * `fast_p_bend_ray_time_a`
Coefficient for arrival time effect by refraction, density dependence.
- static double * `fast_p_bend_ray_time0`
Coefficient for arrival time effect by refraction, altitude dependence.
- static double * `fast_p_rho_rev`
Density steps used in fast interpolation, in order of incr.
- static double `fast_rho_fac`
- static double `etadsn` = 0.000283 * 994186.38 / 1222.656
- static double `observation_level`
Altitude [cm] of observation level.
- static double `obs_level_refidx`

- static double **obs_level_thick**
- static void **init_refraction_tables** ()
Initialize tables needed for atmospheric refraction.
- static void **init_fast_interpolation** ()
An alternate interpolation method (which requires that the table is sufficiently fine-grained and equidistant) has to be initialized first.
- static void **init_corsika_atmosphere** ()
Take the atmospheric profile from CORSIKA built-in functions.
- static void **init_atmosphere_from_text_file** ()
Initialize atmospheric profiles.
- static void **init_atmosphere_from_atmprof** (void)
- static double **sum_log_dev_sq** (double a, double b, double c, int np, double *h, double *t, double *rho)
- static double **atm_exp_fit** (double h1, double h2, double *ap, double *bp, double *cp, double *s0, int *npp)
Fit one atmosphere layer by an exponential density model.
- void **free_atmo_csplines** (void)
- static void **trace_ray_planar** (double emlev, double olev, double za, double step, double *xo, double *to, double *so)
Trace a downward light ray in a planar atmosphere.
- static void **init_common_atmosphere** (void)
Second-stage part of atmospheric profile initialization is common to CORSIKA default profile and tabulated profiles.
- void **atmset_** (int *iatmo, double *obslev)
Set number of atmospheric model profile to be used.
- void **atmnam_** (const char *aname, double *obslev)
Instead of setting the atmospheric profile by number, it gets set by name, also indicated by setting profile number to 99.
- void **atm_init** (AtmProf *aprof)
This variant is not usable from the FORTRAN code side, thus no underscore at the end of the function name.
- double **rhofx_** (double *height)
Density of the atmosphere as a function of altitude.
- double **thickx_** (double *height)
*Atmospheric thickness [g/cm**2] as a function of altitude.*
- double **refim1x_** (double *height)
Index of refraction minus 1 as a function of altitude [cm].
- double **heighx_** (double *thick)
*Altitude [cm] as a function of atmospheric thickness [g/cm**2].*
- void **raybnd_** (double *zem, **cors_dbl_t** *u, **cors_dbl_t** *v, double *w, **cors_dbl_t** *dx, **cors_dbl_t** *dy, **cors_dbl_t** *dt)
Calculate the bending of light due to atmospheric refraction.
- static double **sum_log_dev_sq** (double a, double b, double c, int np, double *h, double *t, double *UNUSED(rho))
Measure of deviation of model layers from tables.
- static double **fn_thick** (double h, int nl, double *hl, double *a, double *b, double *c)
Corresponding to CORSIKA built-in function THICK; only used to show fit results.
- static double **fn_rhof** (double h, int nl, double *hl, double *UNUSED(a), double *b, double *c)
Corresponding to CORSIKA built-in function RHOF; only used to show fit results.
- void **atmfit_** (int *nlp, double *hlay, double *aاتم, double *batm, double *catm)
Fit the tabulated density profile for CORSIKA EGS part.
- void **show_atmo_macros** (void)
- #define **M_PI** (3.14159265358979323846264338327950288)
- /
- #define **WITH_RPOLATOR_CSPLINE** 1 /* Take advantage of cubic splines if we have them available */
- #define **FAST_INTERPOLATION** always

- `#define FAST_INTERPOLATION2` always
- `#define FAST_INTERPOLATION3` always
- `#define WITH_THICKX_DIRECT`
- `#define UNUSED(x) UNUSED_ ## x`
- `#define MAX_PROFILE 120`
- `#define MAX_FAST_PROFILE 40000`
- `#define _XSTR_(s) _STR_(s)`
Expand a macro first and then enclose in string.
- `#define _STR_(s) #s`
Enclose in string without macro expansion.
- `#define SHOW_MACRO(s)`
The following shows the same output after a "#define XYZ" and a "#define XYZ 1" (or compiled with "-DXYZ")
- `typedef double cors_dbl_t`
- `double thick_ (double *height)`
The CORSIKA built-in function for vertical atmospheric thickness (overburden).
- `#define CORSIKA_VERSION 6900`
/
- `typedef float cors_real_t`
*Type for CORSIKA floating point numbers remaining REAL*4.*
- `typedef double cors_dbl_t`
*Type for CORSIKA numbers which are currently REAL*8.*
- `void get_impact_offset (cors_real_t evth[273], cors_dbl_t prmpar[17])`
- `void televt_ (cors_real_t evth[273], cors_dbl_t prmpar[17])`
- `int iact_check_track_ (double *x1, double *y1, double *z1, double *t1, double *e1, double *eta1, double *x2, double *y2, double *z2, double *t2, double *e2, double *eta2, double *amass, double *charge, double *wthin)`
- `#define PRMPAR_SIZE 17`
/

4.1.1 Detailed Description

4.1.2 Macro Definition Documentation

4.1.2.1 _STR_

```
#define _STR_(
    s ) #s
```

4.1.2.2 SHOW_MACRO

```
#define SHOW_MACRO(
    s )
```

Value:

```
if ( strcmp( #s, _XSTR_(s)) != 0 ) \
{ if ( strcmp( "", _XSTR_(s)) != 0 && strcmp( "1", _XSTR_(s)) != 0 ) \
    printf( "    " #s "=" _XSTR_(s) "\n" ); else printf( "    " #s "\n" ); }
```

Other non-empty, not "1" assigned values are shown explicitly. Undefined macros are not shown. Neither are any assigned to itself.

4.1.3 Function Documentation

4.1.3.1 atm_init()

```
void atm_init (
    AtmProf * aprof )
```

It is there for doing the atmosphere initialization like in the other cases but already passing the pre-filled AtmProf (struct mc_atmprof) along, for example after reading CORSIKA data written by IACT/atmo package version 1.60 and up.

Parameters

<i>aprof</i>	Pointer to atmospheric profile table structure.
--------------	---

Returns

(none)

References atmosphere, atmospheric_profile::atmprof_fname, atmospheric_profile::atmprof_id, atmprof_name, observation_level, and atmospheric_profile::obslev.

4.1.3.2 atmfit_()

```
void atmfit_ (
    int * nlp,
    double * hlay,
    double * aatm,
    double * batm,
    double * catm )
```

Fitting of the tabulated atmospheric density profile by piecewise exponential parts as used in CORSIKA. The fits are constrained by fixing the atmospheric thicknesses at the boundaries to the values obtained from the table. Note that not every atmospheric profile can be fitted well by the CORSIKA piecewise models (4*exponential + 1*constant density). In particular, the tropical model is known to be a problem. Setting the boundary heights manually might help. The user is advised to check at least once that the fitted layers represent the tabulated atmosphere sufficiently well, at least at the altitudes most critical for the observations (usually at observation level and near shower maximum but depending on the user's emphasis, this may vary).

Fit all layers (except the uppermost) by exponentials and (if *nlp > 0) try to improve fits by adjusting layer boundaries. The uppermost layer has constant density up to the 'edge' of the atmosphere.

This function may be called from CORSIKA.

Parameters (all pointers since function is called from Fortran):

Parameters

<i>nlp</i>	Number of layers (5, or negative of that if boundaries set manually)
<i>hlay</i>	Vector of layer (lower) boundaries.
<i>aatm,batm,catm</i>	Parameters as used in CORSIKA.

4.1.3.3 atmnam_()

```
void atmnam_ (
    const char * aname,
    double * obslev )
```

It does not actually initialize the atmosphere - this still should be done with atmset, using profile number 99 for that purpose. You need to call atmnam_ before atmset to make use of this feature.

4.1.3.4 atmset_()

```
void atmset_ (
    int * iatmo,
    double * obslev )
```

The atmospheric model is initialized first before the interpolating functions can be used. For efficiency reasons, the functions [rhofx_\(\)](#), [thickx_\(\)](#), ... don't check if the initialisation was done.

This function is called if the 'ATMOSPHERE' keyword is present in the CORSIKA input file.

The function may be called from CORSIKA to initialize the atmospheric model via 'CALL ATMSET(IATMO,OBSLEV)' or such.

Parameters

<i>iatmo</i>	(pointer to) atmospheric profile number; negative for CORSIKA built-in profiles.
<i>obslev</i>	(pointer to) altitude of observation level [cm]

Returns

(none)

4.1.3.5 compact_photon_hit()

```
static int compact_photon_hit (
    struct detstruct * det,
    double x,
    double y,
    double cx,
    double cy,
    double sx,
    double sy,
    double photons,
    double ctime,
    double zem,
    double lambda ) [static]
```

Store a photon bunch in the bunch list for a given telescope. This bunch list is dynamically created and extended as required. This routine is using a more compact format than [photon_hit\(\)](#). This compact format is not appropriate when core distances of telescopes times sine of zenith angle exceed 1000 m.

Parameters

<i>det</i>	pointer to data structure of the detector hit.
<i>x</i>	X position in CORSIKA detection plane [cm]
<i>y</i>	Y position in CORSIKA detection plane [cm]
<i>cx</i>	Direction projection onto X axis
<i>cy</i>	Direction projection onto Y axis
<i>sx</i>	Slope with respect to X axis ($\text{atan}(sx) = \text{acos}(cx)$)
<i>sy</i>	Slope with respect to Y axis ($\text{atan}(sy) = \text{acos}(cy)$)
<i>photons</i>	Bunch size (sizes above 327 cannot be represented)
<i>ctime</i>	Arrival time of bunch in CORSIKA detection plane.
<i>zem</i>	Altitude of emission above sea level [cm]
<i>lambda</i>	Wavelength (0: undetermined, -1: converted to photo-electron)

Returns

0 (O.K.), -1 (failed to save photon bunch)

References `fclose()`, `fopen()`, `INTERNAL_LIMIT`, `NBUNCH`, and `compact_bunch::photons`.

4.1.3.6 extprim_setup()

```
void extprim_setup (
    const char * text )
```

Parameters

<i>text</i>	CORSIKA input card text following the 'IACT EXTPRIM' keywords. Could be parameter values or a file name.
-------------	--

4.1.3.7 extprm_()

```
void extprm_ (
    cors_dbl_t * type,
    cors_dbl_t * eprim,
    double * thetap,
    double * phip )
```

If primaries are to be generated following some special (non-power-law) spectrum, with a mixed composition, or with an angular distribution not achieved by built-in methods, a user-defined implementation of this function can be used to generate a mixture of primaries of any spectrum, composition, and angular distribution.

Be aware that this function would be called before `televt_` and thus (at least for the first shower) before those run parameters not fitting into the run header are known.

Parameters

<i>type</i>	The type number of the primary to be used in CORSIKA (output).
-------------	--

Parameters

<i>eprim</i>	The energy [GeV] to be used for the primary (output).
<i>thetap</i>	The zenith angle [rad] of the primary (output).
<i>phip</i>	The azimuth angle [rad] of the primary in the CORSIKA way (direction of movement from magnetic North counter-clockwise) (output).

4.1.3.8 fn_rhof()

```
static double fn_rhof (
    double h,
    int nl,
    double * hl,
    double * UNUSEDa,
    double * b,
    double * c ) [static]
```

4.1.3.9 fn_thick()

```
static double fn_thick (
    double h,
    int nl,
    double * hl,
    double * a,
    double * b,
    double * c ) [static]
```

References `top_of_atmosphere`.

4.1.3.10 get_impact_offset()

```
void get_impact_offset (
    cors_real_t evth[273],
    cors_dbl_t prmpar[PRMPAR_SIZE] )
```

Get the approximate impact offset of the primary particle due to deflection in the geomagnetic field. The approximation that the curvature radius is large compared to the distance travelled is used. The method is also not very accurate at large zenith angles where curvature of the atmosphere gets important. Therefore a zenith angle cut is applied and showers very close to the horizon are skipped. Only the offset at the lowest detection level is evaluated.

Parameters

<i>evth</i>	CORSIKA event header block
<i>prmpar</i>	CORSIKA primary particle block. We need it to get the particle's relativistic gamma factor (<code>prmpar[2]</code> or <code>prmpar[1]</code> , depending on the CORSIKA version).

Returns

(none)

4.1.3.11 heigh_()

```
double heigh_ (
    double * thickness )
```

References heighx_().

4.1.3.12 heighx_()

```
double heighx_ (
    double * thick )
```

This function can be called from Fortran code as HEIGHX(THICK).

Parameters

<i>thick</i>	(pointer to) atmospheric thickness [g/cm**2]
--------------	--

Returns

altitude [cm]

References bottom_log_thickness, bottom_of_atmosphere, fast_log_thick_fac, num_prof, p_thick, top_layer_↵
2nd_altitude, top_layer_exp_top, top_layer_hscale, top_layer_hscale_rho0_cfacs_inv, top_layer_rho0, top_log_↵
thickness, and top_of_atmosphere.

Here is the caller graph for this function:

**4.1.3.13 iact_param()**

```
static void iact_param (
    const char * text0 ) [static]
```

Parameters

<i>text</i>	Text following the IACT keyword on the input line.
-------------	--

References `getword()`, and `telfil_()`.

4.1.3.14 in_detector()

```
static int in_detector (
    struct detstruct * det,
    double x,
    double y,
    double sx,
    double sy ) [static]
```

Check if a photon bunch (or, similarly, a particle) hits a particular simulated telescope/detector.

Parameters

<i>x</i>	X position of photon position in CORSIKA detection level [cm]
<i>y</i>	Y position of photon position in CORSIKA detection level [cm]
<i>sx</i>	Slope of photon direction in X/Z plane.
<i>sy</i>	Slope of photon direction in Y/Z plane.

Returns

0 (does not hit), 1 (does hit)

4.1.3.15 init_atmosphere_from_text_file()

```
static void init_atmosphere_from_text_file ( ) [static]
```

Internal function for initialising both external and CORSIKA built-in atmospheric profiles. If any CORSIKA built-in profile should be used, it simply calls `init_corsika_atmosphere()`.

Otherwise, atmospheric models are read in from text-format tables. The supplied models 1-6 are based on output of the MODTRAN program. For the interpolation of relevant parameters (density, thickness, index of refraction, ...) all parameters are transformed such that linear interpolation can be easily used.

4.1.3.16 init_corsika_atmosphere()

```
static void init_corsika_atmosphere ( ) [static]
```

For use of the refraction bending corrections together with the CORSIKA built-in atmospheres, the atmosphere tables are constructed from the CORSIKA RHO and THICK functions. Note that the refraction index in this case is without taking the effect of water vapour into account.

References `atmosphere`, `heigh_()`, `num_prof`, and `top_of_atmosphere`.

4.1.3.17 init_refraction_tables()

```
static void init_refraction_tables ( ) [static]
```

Initialize the correction tables used for the refraction bending of the light paths. It is called once after the atmospheric profile has been defined.

4.1.3.18 Nint_f()

```
static int Nint_f (
    double x ) [static]
```

4.1.3.19 photon_hit()

```
static int photon_hit (
    struct detstruct * det,
    double x,
    double y,
    double cx,
    double cy,
    double sx,
    double sy,
    double photons,
    double ctime,
    double zem,
    double lambda ) [static]
```

Store a photon bunch in the bunch list for a given telescope. It is kept in memory or temporary disk storage until the end of the event. This way, photon bunches are sorted by telescope. This bunch list is dynamically created and extended as required.

Parameters

<i>det</i>	pointer to data structure of the detector hit.
<i>x</i>	X position in CORSIKA detection plane [cm]
<i>y</i>	Y position in CORSIKA detection plane [cm]
<i>cx</i>	Direction projection onto X axis
<i>cy</i>	Direction projection onto Y axis
<i>sx</i>	Slope with respect to X axis ($\text{atan}(sx) = \text{acos}(cx)$)
<i>sy</i>	Slope with respect to Y axis ($\text{atan}(sy) = \text{acos}(cy)$)
<i>photons</i>	Bunch size
<i>ctime</i>	Arrival time of bunch in CORSIKA detection plane.
<i>zem</i>	Altitude of emission above sea level [cm]
<i>lambda</i>	Wavelength (0: undetermined, -1: converted to photo-electron)

Returns

0 (O.K.), -1 (failed to save photon bunch)

Note: With the EXTENDED_TELOUT every second call would have data of the emitting particle: the mass in *cx*, the charge in *cy*, the energy in *photons*, the time of emission in *zem*, and 9999 in *lambda*.

4.1.3.20 raybnd_()

```
void raybnd_ (
    double * zem,
    cors_dbl_t * u,
    cors_dbl_t * v,
    double * w,
    cors_dbl_t * dx,
    cors_dbl_t * dy,
    cors_dbl_t * dt )
```

Path of light through the atmosphere including the bending by refraction. This function assumes a plane-parallel atmosphere. Coefficients for corrections from straight-line propagation to refraction-bent path are numerically evaluated when the atmospheric model is defined. Note that while the former mix of double/float data types may appear odd, it was determined by the variables present in older CORSIKA to save conversions. With CORSIKA 6.0 all parameters are of double type.

This function may be called from FORTRAN as CALL RAYBND(ZEM,U,V,W,DX,DY,DT)

Parameters

<i>zem</i>	Altitude of emission above sea level [cm]
<i>u</i>	Initial/Final direction cosine along X axis (updated)
<i>v</i>	Initial/Final direction cosine along Y axis (updated)
<i>w</i>	Initial/Final direction cosine along Z axis, positive is downwards (updated)
<i>dx</i>	Position in CORSIKA detection plane [cm] (updated)
<i>dy</i>	Position in CORSIKA detection plane [cm] (updated)
<i>dt</i>	Time of photon [ns]. Input: emission time. Output: time of arrival in CORSIKA detection plane.

References observation_level, and rhofx_().

4.1.3.21 refidx_()

```
double refidx_ (
    double * height )
```

This function can be called from Fortran code as REFIDX(HEIGHT).

Parameters

<i>height</i>	(pointer to) altitude [cm]
---------------	----------------------------

Returns

index of refraction

4.1.3.22 refim1x_()

```
double refim1x_ (
    double * height )
```

This function can be called from Fortran code as REFIM1X(HEIGHT). This part has been split off from refidx_ in order to be able to access values small compared to 1.0 without big (relative) rounding errors.

Parameters

<i>height</i>	(pointer to) altitude [cm]
---------------	----------------------------

Returns

index of refraction minus one

References bottom_of_atmosphere, fast_p_alt, fast_p_n1, p_log_n1, rpol_cspline(), and top_of_atmosphere.

Here is the caller graph for this function:

**4.1.3.23 rhof_()**

```
double rhof_ (  
    double * height )
```

4.1.3.24 rhofx_()

```
double rhofx_ (  
    double * height )
```

This function can be called from Fortran code as RHOFX(HEIGHT).

Parameters

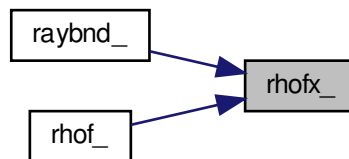
<i>height</i>	(pointer to) altitude [cm]
---------------	----------------------------

Returns

density [g/cm**3]

References bottom_of_atmosphere, fast_p_alt, fast_p_rho, p_rho, rpol_cspline(), and top_of_atmosphere.

Here is the caller graph for this function:



4.1.3.25 `rndm()`

```
static double rndm (
    int dummy ) [static]
```

4.1.3.26 `sample_offset()`

```
void sample_offset (
    const char * sampling_fname,
    double core_range,
    double theta,
    double phi,
    double thetaref,
    double phiref,
    double offax,
    double E,
    int primary,
    double * xoff,
    double * yoff,
    double * sampling_area )
```

Parameters

<i>sampling_fname</i>	Name of file with parameters, to be read on first call.
<i>core_range</i>	Maximum core distance as used in data format check [cm]. If not obeying this maximum distance, make sure to switch on the long data format manually.
<i>theta</i>	Zenith angle [radians]
<i>phi</i>	Shower azimuth angle in CORSIKA angle convention [radians].
<i>thetaref</i>	Reference zenith angle (e.g. of VIEWCONE centre) [radians].
<i>phiref</i>	Reference azimuth angle (e.g. of VIEWCONE centre) [radians].
<i>offax</i>	Angle between central direction (typically VIEWCONE centre) and the direction of the current primary [radians].
<i>E</i>	Energy of primary particle [GeV]
<i>primary</i>	Primary particle ID.
<i>xoff</i>	X offset [cm] to be generated.
<i>yoff</i>	Y offset [cm] to be generated.
<i>sampling_area</i>	Area weight of the generated sample (normalized to $\text{Pi} \cdot \text{core_range}^2$) [cm^2].

4.1.3.27 set_random_systems()

```
static int set_random_systems (
    double theta,
    double phi,
    double thetaref,
    double phiref,
    double offax,
    double E,
    int primary,
    int volflag ) [static]
```

The area containing the detectors is sub-divided into a rectangular grid and each detector with a (potential) intersection with a grid element is marked for that grid element. A detector can be marked for several grid elements unless completely inside one element. Checks which detector(s) is/are hit by a photon bunch (or, similarly, by a particle) is thus reduced to check only the detectors marked for the grid element which is hit by the photon bunch (or particle). The grid should be sufficiently fine-grained that there are usually not much more than one detector per element but finer graining than the detector sizes makes no sense.

Parameters

<i>theta</i>	Zenith angle of the shower following [radians].
<i>phi</i>	Shower azimuth angle in CORSIKA angle convention [radians].
<i>thetaref</i>	Reference zenith angle (e.g. of VIEWCONE centre) [radians].
<i>phiref</i>	Reference azimuth angle (e.g. of VIEWCONE centre) [radians].
<i>offax</i>	Angle between central direction (typically VIEWCONE centre) and the direction of the current primary [radians].
<i>E</i>	Primary particle energy in GeV (may be used in importance sampling).
<i>primary</i>	Primary particle ID (may be used in importance sampling).
<i>volflag</i>	Set to 1 if CORSIKA was compiled with VOLUMEDET option, 0 otherwise.

Returns

0 (O.K.), -1 (error)

4.1.3.28 set_system_displacement()

```
void set_system_displacement (
    double dx,
    double dy )
```

The given displacement is added on top of generated core offsets. This function is not used with CORSIKA.

References `xdisplaced`.

4.1.3.29 telasu_()

```
void telasu_ (
    int * n,
    cors_dbl_t * dx,
    cors_dbl_t * dy )
```

Set up how many times the telescope system should be randomly scattered within a given area. Thus each telescope system (array) will see the same shower but at random offsets. Each shower is thus effectively used several times. This function is called according to the CSCAT keyword in the CORSIKA input file.

Parameters

<i>n</i>	The number of telescope systems
<i>dx</i>	Core range radius (if <i>dy</i> =0) or core x range
<i>dy</i>	Core y range (non-zero for ractangular, 0 for circular)

Returns

(none)

References `core_range`, and `core_range1`.

4.1.3.30 telend_()

```
void telend_ (
    cors_real_t evte[273] )
```

Write out all recorded photon bunches.

End of an event: write all stored photon bunches to the output data file, and the CORSIKA event end block as well.

Parameters

<i>evte</i>	CORSIKA event end block
-------------	-------------------------

Returns

(none)

4.1.3.31 televt_()

```
void televt_ (
    cors_real_t evth[273],
    cors_dbl_t prmpar[PRMPAR_SIZE] )
```

Save event parameters.

Start of new event: get parameters from CORSIKA event header block, create randomly scattered telescope systems in given area, and write their positions as well as the CORSIKA block to the data file.

Parameters

<i>evth</i>	CORSIKA event header block
<i>prmpar</i>	CORSIKA primary particle block

Returns

(none)

4.1.3.32 telfil_()

```
void telfil_ (
    const char * name0 )
```

This function is called when the 'TELFIL' keyword is present in the CORSIKA input file.

```
* The 'file name' parsed is actually decoded further:
*   Apart from the leading '+' or '|' or '+|' the TELFIL argument
*   may contain further bells and whistles:
*   If the supplied file name contains colons, they are assumed to
*   separate appended numbers with the following meaning:
*   #1: number of events for which the photons per telescope are shown
*   #2: number of events for which energy, direction etc. are shown
*   #3: every so often an event is shown (e.g. 10 -> every tenth event).
*   #4: every so often the event number is shown even if #1 and #2 ran out.
*   #5: offset for #4 (#4=100, #5=1: show events 1, 101, 201, ...)
*   #6: the maximum number of photon bunches before using external storage
*   #7: the maximum size of the output buffer in Megabytes.
*   Example: name = "iact.dat:5:15:10"
*   name becomes "iact.dat"
*   5 events are fully shown
*   15 events have energy etc. shown
*   Every tenth event is shown, i.e. 10,20,30,40,50 are fully shown
*   and events number 60,...,150 have their energies etc. shown.
*   After that every shower with event number divideable by 1000 is shown.
*   Note: No spaces inbetween! CORSIKA input processing truncates at blanks.
*
*   Instead of adding these numbers to the TELFIL parameter you can use
*   separate IACT input parameters - which may be less confusing:
*   TELFIL +iact.dat:5:15:10:1000:1:100000:200
*   is equivalent to
*   TELFIL +iact.dat
*   IACT print_events 5 15 10 1000 1
*   IACT internal_bunches 100000
*   IACT io_buffer 200MB
*
```

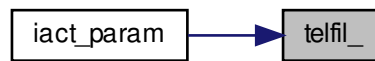
Parameters

<i>name</i>	Output file name. Note: A leading '+' means: use non-compact format A leading ' ' (perhaps after '+') means that the name will not be interpreted as the name of a data file but of a program to which the 'eventio' data stream will be piped (i.e. that program should read the data from its standard input. Any command-line options to that program can be defined via an 'IACT telopt' line.
-------------	--

Returns

(none)

Here is the caller graph for this function:



4.1.3.33 telfil_()

```

void telfil_ (
    int * itel,
    double * x,
    double * y,
    double * z,
    double * r,
    int * exists )
  
```

Parameters

<i>itel</i>	number of telescope in question
<i>x,y,z</i>	telescope position [cm]
<i>r</i>	radius of fiducial volume [cm]
<i>exists</i>	telescope exists

References `ntel`, and `xtel`.

4.1.3.34 tellng_()

```

void tellng_ (
    int * type,
    double * data,
    int * ndim,
    int * np,
    int * nthick,
    double * thickstep )
  
```

Write several kinds of vertical distributions to the output. These are kinds of histograms as a function of atmospheric depth. In CORSIKA, these are generally referred to as 'longitudinal' distributions.

```

*   There are three types of distributions:
*       type 1: particle distributions for
*               gammas, positrons, electrons, mu+, mu-,
*               hadrons, all charged, nuclei, Cherenkov photons.
*       type 2: energy distributions (with energies in GeV) for
*               gammas, positrons, electrons, mu+, mu-,
*               hadrons, all charged, nuclei, sum of all.
  
```

```

*      type 3: energy deposits (in GeV) for
*              gammas, e.m. ionisation, cut of e.m. particles,
*              muon ionisation, muon cut, hadron ionisation,
*              hadron cut, neutrinos, sum of all.
*              ('cut' accounting for low-energy particles dropped)
*

```

Note: Corsika can be built with three options concerning the vertical profile of Cherenkov light: default = emission profile, INTCLONG = integrated light profile, NOCLONG = no Cherenkov profiles at all. If you know which kind you are using, you are best off by defining it for compilation of this file (either -DINTEGRATED_LONG_DIST, -DEMISSION_LONG_DIST, or -DNO_LONG_DIST). By default, a run-time detection is attempted which should work well with some 99.99% of all air showers but may fail in some cases like non-interacting muons as primary particles etc.

If eventio functionality is disabled this function does nothing.

Parameters

<i>type</i>	see above
<i>data</i>	set of (usually 9) distributions
<i>ndim</i>	maximum number of entries per distribution
<i>np</i>	number of distributions (usually 9)
<i>nthick</i>	number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
<i>thickstep</i>	step size in g/cm**2

Returns

(none)

4.1.3.35 tellni_()

```

void tellni_ (
    const char * line,
    int * llength )

```

Add a CORSIKA input line to a linked list of strings which will be written to the output file in eventio format right after the run header.

Parameters

<i>line</i>	input line (not terminated)
<i>llength</i>	maximum length of input lines (132 usually)

4.1.3.36 telout_()

```

int telout_ (
    cors_dbl_t * bsize,
    cors_dbl_t * wt,
    cors_dbl_t * px,
    cors_dbl_t * py,

```



```

cors_dbl_t * pu,
cors_dbl_t * pv,
cors_dbl_t * ctime,
cors_dbl_t * zem,
cors_dbl_t * lambda )

```

A bunch of photons from CORSIKA is checked if they hit a telescope and in this case it is stored (in memory). This routine can alternatively trigger that the photon bunch is written by CORSIKA in its usual photons file.

Note that this function should only be called for downward photons as there is no parameter that could indicate upwards photons.

The interface to this function can be modified by defining EXTENDED_TELOUT. Doing so requires to have a CORSIKA version with support for the IACTEXT option, and to actually activate that option. That could be useful when adding your own code to create some nice graphs or statistics that requires to know the emitting particle and its energy but would be of little help for normal use. Inconsistent usage of EXTENDED_TELOUT here and IACTEXT in CORSIKA will most likely lead to a crash.

Parameters

<i>bsize</i>	Number of photons (can be fraction of one)
<i>wt</i>	Weight (if thinning option is active)
<i>px</i>	x position in detection level plane
<i>py</i>	y position in detection level plane
<i>pu</i>	x direction cosine
<i>pv</i>	y direction cosine
<i>ctime</i>	arrival time in plane after first interaction
<i>zem</i>	height of emission above sea level
<i>lambda</i>	0. (if wavelength undetermined) or wavelength [nm]. If $\lambda < 0$, photons are already converted to photo-electrons (p.e.), i.e. we have p.e. bunches.
<i>temis</i>	Time of photon emission (only if CORSIKA extracted with IACTEXT option and this code compiled with EXTENDED_TELOUT defined).
<i>penergy</i>	Energy of emitting particle (under conditions as temis).
<i>amass</i>	Mass of emitting particle (under conditions as temis).
<i>charge</i>	Charge of emitting particle (under conditions as temis).

Returns

0 (no output to old-style CORSIKA file needed) 2 (detector hit but no eventio interface available or output should go to CORSIKA file anyway)

References impact_offset, compact_bunch::lambda, and compact_bunch::y.

4.1.3.37 telrne_()

```

void telrne_ (
    cors_real_t rune[273] )

```

Parameters

<i>rune</i>	CORSIKA run end block
-------------	-----------------------

4.1.3.38 telrnh_()

```
void telrnh_ (
    cors_real_t runh[273] )
```

Get relevant parameters from CORSIKA run header block and write run header block to the data output file.

Parameters

<i>runh</i>	CORSIKA run header block
-------------	--------------------------

Returns

(none)

4.1.3.39 telset_()

```
void telset_ (
    cors_dbl_t * x,
    cors_dbl_t * y,
    cors_dbl_t * z,
    cors_dbl_t * r )
```

Set up another telescope for the simulated telescope system. No details of a telescope need to be known except for a fiducial sphere enclosing the relevant optics. Actually, the detector could as well be a non-imaging device.

This function is called for each TELESCOPE keyword in the CORSIKA input file.

Parameters

<i>x</i>	X position [cm]
<i>y</i>	Y position [cm]
<i>z</i>	Z position [cm]
<i>r</i>	radius [cm] within which the telescope is fully contained

Returns

(none)

References MAX_ARRAY_SIZE, ntel, and xtel.

4.1.3.40 telshw_()

```
void telshw_ (
    void )
```

This function is called by CORSIKA after the input file is read.

References output_fname.

4.1.3.41 `telsmp_()`

```
void telsmp_ (
    const char * name )
```

@short Set the file name with parameters for importance sampling.

Note that the TELSAMPLE parameter is not processed by CORSIKA itself and thus has to be specified through configuration lines like

```
IACT TELSAMPLE filename
*(IACT) TELSAMPLE filename
```

where the first form requires a CORSIKA patch and the second would work without that patch (but then only with uppercase file names).

References `sampling_fname`.

4.1.3.42 `thick_()`

```
double thick_ (
    double * height )
```

References `thickx_()`.

4.1.3.43 `thickx_()`

```
double thickx_ (
    double * height )
```

This function can be called from Fortran code as THICKX(HEIGHT).

Parameters

<i>height</i>	(pointer to) altitude [cm]
---------------	----------------------------

Returns

thickness [g/cm**2]

References `bottom_of_atmosphere`, `fast_p_alt`, `fast_p_thick`, `num_prof`, `p_alt`, `p_thick`, `rpol_cspline()`, `top_layer↵`
`_2nd_altitude`, `top_layer_exp_top`, `top_layer_hscale`, `top_layer_hscale_rho0_cfacs`, `top_layer_rho0`, and `top_of_↵`
`atmosphere`.

Here is the caller graph for this function:



4.1.3.44 trace_ray_planar()

```
static void trace_ray_planar (  
    double emlev,  
    double olev,  
    double za,  
    double step,  
    double * xo,  
    double * to,  
    double * so ) [static]
```

Parameters

<i>emlev</i>	Emission level altitude [cm]
<i>olev</i>	Observation level altitude [cm] (below emlev)
<i>za</i>	Zenith angle (positive downwards)
<i>step</i>	Step length [cm]
<i>xo</i>	Arrival position at observation level [cm] (returned only), positive value expected for downward bending.
<i>to</i>	Travel time to observation level [ns] (returned only).
<i>so</i>	Path length travelled [cm] (returned only).

References `refidx_()`.

4.1.4 Variable Documentation

4.1.4.1 atmosphere

```
int atmosphere
```

4.1.4.2 atmprof_name

```
char* atmprof_name = NULL [static]
```

4.1.4.3 core_range

```
double core_range [static]
```

4.1.4.4 core_range1

```
double core_range1 [static]
```

4.1.4.5 corsika_version

```
int corsika_version = (CORSIKA_VERSION)
```

4.1.4.6 dmax

```
double dmax = 0. [static]
```

distance of telescopes in (x,y)

4.1.4.7 fast_p_rho_rev

```
double* fast_p_rho_rev [static]
```

density

4.1.4.8 impact_correction

```
int impact_correction = 1 [static]
```

4.1.4.9 max_internal_bunches

```
int max_internal_bunches = INTERNAL_LIMIT [static]
```

4.1.4.10 max_io_buffer

```
size_t max_io_buffer = MAX_IO_BUFFER [static]
```

4.1.4.11 num_prof

```
int num_prof [static]
```

4.1.4.12 output_fname

```
char* output_fname = NULL [static]
```

4.1.4.13 p_alt

```
double p_alt[MAX_PROFILE] [static]
```

4.1.4.14 p_alt_rev

```
double p_alt_rev[MAX_PROFILE] [static]
```

4.1.4.15 p_log_n1

```
double p_log_n1[MAX_PROFILE] [static]
```

4.1.4.16 rmax

```
double rmax = 0. [static]
```

radius of telescopes

4.1.4.17 sampling_fname

```
char* sampling_fname [static]
```

4.1.4.18 theta_central

```
double theta_central [static]
```

4.1.4.19 top_layer_2nd_altitude

```
double top_layer_2nd_altitude [static]
```

4.1.4.20 top_layer_cfac

```
double top_layer_cfac [static]
```

4.1.4.21 top_layer_hscale

```
double top_layer_hscale [static]
```

4.1.4.22 top_layer_hscale_rho0_cfacs_inv

```
double top_layer_hscale_rho0_cfacs_inv [static]
```

$1/(\text{top_layer_rho0} * \text{top_layer_cfacs} * \text{top_layer_hscale})$

4.1.4.23 top_layer_rho0

```
double top_layer_rho0 [static]
```

4.1.4.24 top_log_thickness

```
double top_log_thickness [static]
```

for fast interpolation in reverse direction (thickness to height).

4.1.4.25 top_of_atm_table

```
double top_of_atm_table [static]
```

4.1.4.26 xtel

```
double xtel[MAX_ARRAY_SIZE] [static]
```

4.2 The fake_corsika program

Data Structures

- struct `photon_bunch`
- struct `telpos_struct`

Typedefs

- typedef struct `photon_bunch` **Bunch**
- typedef struct `telpos_struct` **TelPos**

Functions

- void **addhist** (const char *txt)
- double `heigh_` (double *thick)
The CORSIKA built-in function for the height as a function of overburden.
- int **main** ()
- double `rhof_` (double *height)
The CORSIKA built-in density lookup function.
- void **rmmard_** (double *a, int *b, int *c)
- double `thick_` (double *height)
The CORSIKA built-in function for vertical atmospheric thickness (overburden).

Variables

- static **Bunch** **bunch**
- `cors_dbl_t` **cscat1** = 0.
- `cors_dbl_t` **cscat2** = 0.
- static `cors_real_t` **evte** [273]
- static `cors_real_t` **evth** [273]
- char **fname** [1024]
- int **iatmo** = 1
- int **ndim** =15
- int **nprof** =9
- static int **nscat** = 1
- int **nshow** = 0
- int **nthick** =10
- double **obslev** = 2000e2
- static `cors_dbl_t` **prmpar** [**PRMPAR_SIZE**]
- double **profile** [9][15]
- int **ptype** =1
- double **rtel** = 10e2
- static `cors_real_t` **rune** [273]
- static `cors_real_t` **runh** [273]
- static **TelPos** **telpos**
- double **thickstep** =100

4.2.1 Detailed Description

4.2.2 Function Documentation

4.2.2.1 heigh_()

```
double heigh_ (  
    double * thick )
```

4.2.2.2 rhof_()

```
double rhof_ (  
    double * height )
```

4.2.2.3 thick_()

```
double thick_ (  
    double * height )
```

References thickx_().

4.3 The listio program

Functions

- `int main (int argc, char **argv)`
Main function.

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 main()

```
int main (  
    int argc,  
    char ** argv )
```

The main function of the listio program.

4.4 The read_iact program

Functions

- int `main` (int argc, char **argv)
Main program.
- int `my_print_hess_mc_phot` (IO_BUFFER *iobuf)
Print Monte Carlo photons and photo-electrons.
- void `syntax` (void)

4.4.1 Detailed Description

4.4.2 Function Documentation

4.4.2.1 `main()`

```
int main (  
    int argc,  
    char ** argv )
```

Main program function of read_hess.c program.

4.5 The sim_skeleton program

Data Structures

- struct `camera_electronics`
Parameters of the electronics of a telescope.
- struct `mc_options`
Options of the simulation passed through to low-level functions.
- struct `mc_run`
Basic parameters of the CORSIKA run.
- struct `pm_camera`
Parameters of a telescope camera (pixels, ...)
- struct `simulated_shower_parameters`
Basic parameters of a simulated shower.
- struct `telescope_array`
Description of telescope position, array offsets and shower parameters.
- struct `telescope_optics`
Parameters describing the telescope optics.

Macros

- `#define MAX_ARRAY 100`
The largest no.
- `#define MAX_BUNCHES 2500000`
Change the following limits as appropriate <<<<
- `#define MAX_PHOTOELECTRONS 1000000` */** The largest number of photo-electrons. */*
- `#define MAX_PIXELS 1024`
The largest no.
- `#define MAX_TEL 16`
The largest no.
- `#define Nair(hkm) (1.+0.0002814*exp(-0.0947982*(hkm)-0.00134614*(hkm)*(hkm)))`
Refraction index of air as a function of height in km (0km<=h<=8km)

Functions

- double **atmospheric_transmission** (int iwl, double zem, double airmass)
- void **atmset_** (int *iatmo, double *obslev)
Set number of atmospheric model profile to be used.
- double **find_max_pos** (double *y, int n)
- double **heigh_** (double *x)
The CORSIKA built-in function for the height as a function of overburden.
- double **line_point_distance** (double x1, double y1, double z1, double cx, double cy, double cz, double x, double y, double z)
Distance between a straight line and a point in space.
- int **main** (int argc, char **argv)
Main program of Cherenkov telescope simulation.
- double **RandFlat** (void)
- double **rhof_** (double *h)
The CORSIKA built-in density lookup function.
- double **thick_** (double *h)
The CORSIKA built-in function for vertical atmospheric thickness (overburden).

Variables

- static double **airlightspeed** = 29.9792458/1.0002256
- struct [linked_string](#) **corsika_inputs**

4.5.1 Detailed Description

4.5.2 Macro Definition Documentation

4.5.2.1 MAX_ARRAY

```
#define MAX_ARRAY 100
```

of arrays to be handled

4.5.2.2 MAX_BUNCHES

```
#define MAX_BUNCHES 2500000
```

The largest no. of bunches that can be handled.

4.5.2.3 MAX_PIXELS

```
#define MAX_PIXELS 1024
```

of pixels per camers

4.5.2.4 MAX_TEL

```
#define MAX_TEL 16
```

of telescopes/array.

4.5.3 Function Documentation

4.5.3.1 atmset_()

```
void atmset_ (
    int * iatmo,
    double * obslev )
```

The atmospheric model is initialized first before the interpolating functions can be used. For efficiency reasons, the functions [rho_{fx}_\(\)](#), [thickx_\(\)](#), ... don't check if the initialisation was done.

This function is called if the 'ATMOSPHERE' keyword is present in the CORSIKA input file.

The function may be called from CORSIKA to initialize the atmospheric model via 'CALL ATMSET(IATMO,OBSLEV)' or such.

Parameters

<i>iatmo</i>	(pointer to) atmospheric profile number; negative for CORSIKA built-in profiles.
<i>obslev</i>	(pointer to) altitude of observation level [cm]

Returns

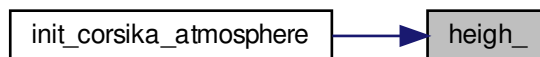
(none)

4.5.3.2 heigh_()

```
double heigh_ (  
    double * thick )
```

References heighx_().

Here is the caller graph for this function:

**4.5.3.3 line_point_distance()**

```
double line_point_distance (  
    double x1,  
    double y1,  
    double z1,  
    double cx,  
    double cy,  
    double cz,  
    double x,  
    double y,  
    double z )
```

Parameters

<i>x1,y1,z1</i>	reference point on the line
<i>cx,cy,cz</i>	direction cosines of the line
<i>x,y,z</i>	point in space

Returns

distance

4.5.3.4 rhof_()

```
double rhof_ (  
    double * height )
```

References rhofx_().

4.5.3.5 thick_()

```
double thick_ (  
    double * height )
```

4.6 The testio program

Data Structures

- struct `test_struct`

Typedefs

- typedef struct `test_struct` `TEST_DATA`

Functions

- int `datacmp` (`TEST_DATA` *data1, `TEST_DATA` *data2)
Compare elements of test data structures.
- int `main` (int argc, char **argv)
Main function for I/O test program.
- int `read_test1` (`TEST_DATA` *data, `IO_BUFFER` *iobuf)
Read test data with single-element functions.
- int `read_test2` (`TEST_DATA` *data, `IO_BUFFER` *iobuf)
Read test data with vector functions as far as possible.
- int `read_test3` (`TEST_DATA` *data, `IO_BUFFER` *iobuf)
Read test data as a nested tree.
- int `read_test_lrg` (`IO_BUFFER` *iobuf)
Read large data block.
- void `syntax` (const char *prg)
Replacement for function missing on OS-9.
- int `write_test1` (`TEST_DATA` *data, `IO_BUFFER` *iobuf)
Write test data with single-element functions.
- int `write_test2` (`TEST_DATA` *data, `IO_BUFFER` *iobuf)
Write test data with vector functions as far as possible.
- int `write_test3` (`TEST_DATA` *data, `IO_BUFFER` *iobuf)
Write test data in nested items.
- int `write_test_lrg` (`IO_BUFFER` *iobuf)
Write large data block.

Variables

- static int `care_int`
- static int `care_long`
- static int `care_short`

4.6.1 Detailed Description

4.6.2 Function Documentation

4.6.2.1 datacmp()

```
int datacmp (  
    TEST_DATA * data1,  
    TEST_DATA * data2 )
```

Compare elements of test data structures with the accuracy relevant to the I/O package.

Parameters

<i>data1</i>	first data structure
<i>data2</i>	second data structure

Returns

0 (something did not match), 1 (O.K.)

4.6.2.2 main()

```
int main (
    int argc,
    char ** argv )
```

First writes a test data structure with the vector functions, then the same data structure with the single-element functions. The output file is then closed and reopened for reading. The first structure is then read with the single-element functions and the second with the vector functions (i.e. the other way as done for writing). The data from the file is compared with the original data, taking the relevant accuracy into account. Note that if an 'int' variable is written via 'put_short()' and then read again via 'get_short()' not only the upper two bytes (on a 32-bit machine) are lost but also the sign bit is propagated from bit 15 to the upper 16 bits. Similarly, if a 'long' variable is written via 'put_long()' and later read via 'get_long()' on a 64-bit-machine, not only the upper 4 bytes are lost but also the sign in bit 31 is propagated to the upper 32 bits.

4.6.2.3 read_test1()

```
int read_test1 (
    TEST_DATA * data,
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for [get_item_end\(\)](#))

4.6.2.4 read_test2()

```
int read_test2 (
    TEST_DATA * data,
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for [get_item_end\(\)](#))

4.6.2.5 read_test3()

```
int read_test3 (
    TEST_DATA * data,
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for [get_item_end\(\)](#))

4.6.2.6 read_test_lrg()

```
int read_test_lrg (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	Pointer to I/O buffer
--------------	-----------------------

Returns

0 (ok), <0 (error as for [put_item_end\(\)](#))

4.6.2.7 write_test1()

```
int write_test1 (
    TEST_DATA * data,
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (O.K.), <0 (error as for [put_item_end\(\)](#))

4.6.2.8 write_test2()

```
int write_test2 (
    TEST_DATA * data,
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for [put_item_end\(\)](#))

4.6.2.9 write_test3()

```
int write_test3 (
    TEST_DATA * data,
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for [put_item_end\(\)](#))

4.6.2.10 write_test_lrg()

```
int write_test_lrg (
    IO_BUFFER * iobuf )
```

Parameters

<i>data</i>	Pointer to test data structure
<i>iobuf</i>	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for [put_item_end\(\)](#))

5 Data Structure Documentation

5.1 `_struct_IO_BUFFER` Struct Reference

The `IO_BUFFER` structure contains all data needed the manage the stuff.

```
#include <io_basic.h>
```

Data Fields

- `int aux_count`
May be used for dedicated buffers.
- `unsigned char * buffer`
Pointer to allocated data space.
- `long buflen`
Usable length of data space.
- `int byte_order`
Set if block is not in internal byte order.
- `BYTE * data`
Position for next get.../put...
- `int data_pending`
Set to 1 when header is read but not the data.
- `int extended`
Set to 1 if you want to use the extension field always.
- `FILE * input_file`
For use of stream I/O for input.
- `int input_fileno`
For use of read() function for input.
- `int is_allocated`
Indicates if buffer is allocated by eventio.
- `int item_extension [MAX_IO_ITEM_LEVEL]`
Where the extension field was used.
- `long item_length [MAX_IO_ITEM_LEVEL]`
Length of each level of items.
- `int item_level`
Current level of nesting of items.
- `long item_start_offset [MAX_IO_ITEM_LEVEL]`
Where the item starts in buffer.
- `long max_length`
The maximum length for extending the buffer.
- `long min_length`
The initial and minimum length of the buffer.
- `int msg_ext`
1 if buffer extension should be reported, 0 if not
- `FILE * output_file`
For use of stream I/O for output.
- `int output_fileno`
For use of write() function for output.
- `long r_remaining`
- `int regular`

- 1 if a regular file, 0 not known, -1 not regular*
- long `sub_item_length` [MAX_IO_ITEM_LEVEL]
Length of its sub-items.
- int `sync_err_count`
Count of synchronization errors.
- int `sync_err_max`
Maximum accepted number of synchronisation errors.
- int(* `user_function`)(unsigned char *, long, int)
For use of special type of I/O.
- long `w_remaining`
Byte available for reading/writing.

5.1.1 Detailed Description

5.1.2 Field Documentation

5.1.2.1 `buffer`

```
unsigned char* _struct_IO_BUFFER::buffer
```

5.1.2.2 `buflen`

```
long _struct_IO_BUFFER::buflen
```

5.1.2.3 `byte_order`

```
int _struct_IO_BUFFER::byte_order
```

5.1.2.4 `data`

```
BYTE* _struct_IO_BUFFER::data
```

5.1.2.5 `extended`

```
int _struct_IO_BUFFER::extended
```

5.1.2.6 `input_file`

```
FILE* _struct_IO_BUFFER::input_file
```

5.1.2.7 `input_fileno`

```
int _struct_IO_BUFFER::input_fileno
```

5.1.2.8 `is_allocated`

```
int _struct_IO_BUFFER::is_allocated
```

It is 1 if buffer is allocated by eventio, 0 if buffer provided by user function (in which case the user should call `allocate_io_buffer` with the appropriate size; then the buffer always allocated in [allocate_io_buffer\(\)](#) must be freed by the user function, replaced by its external buffer, and finally `is_allocated` set to 0).

5.1.2.9 `item_extension`

```
int _struct_IO_BUFFER::item_extension[MAX_IO_ITEM_LEVEL]
```

5.1.2.10 `item_level`

```
int _struct_IO_BUFFER::item_level
```

5.1.2.11 `item_start_offset`

```
long _struct_IO_BUFFER::item_start_offset[MAX_IO_ITEM_LEVEL]
```

5.1.2.12 `output_file`

```
FILE* _struct_IO_BUFFER::output_file
```

5.1.2.13 `output_fileno`

```
int _struct_IO_BUFFER::output_fileno
```

5.1.2.14 sync_err_count

```
int _struct_IO_BUFFER::sync_err_count
```

5.1.2.15 sync_err_max

```
int _struct_IO_BUFFER::sync_err_max
```

5.1.2.16 user_function

```
int(* _struct_IO_BUFFER::user_function) (unsigned char *, long, int)
```

5.1.2.17 w_remaining

```
long _struct_IO_BUFFER::w_remaining
```

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

- [io_basic.h](#)

5.2 _struct_IO_ITEM_HEADER Struct Reference

An IO_ITEM_HEADER is to access header info for an I/O block and as a handle to the I/O buffer.

```
#include <io_basic.h>
```

Data Fields

- int [can_search](#)
Set to 1 if I/O block consist of sub-blocks only.
- long [ident](#)
Identity number.
- size_t [length](#)
Length of data field, for information only.
- int [level](#)
Tells how many levels deep we are nested now.
- unsigned long [type](#)
The type number telling the type of I/O block.
- int [use_extension](#)
Non-zero if the extension header field should be used.
- int [user_flag](#)
One more bit in the header available for user data.
- unsigned [version](#)
The version number used for the block.

5.2.1 Detailed Description

5.2.2 Field Documentation

5.2.2.1 `can_search`

```
int _struct_IO_ITEM_HEADER::can_search
```

5.2.2.2 `ident`

```
long _struct_IO_ITEM_HEADER::ident
```

5.2.2.3 `length`

```
size_t _struct_IO_ITEM_HEADER::length
```

5.2.2.4 `level`

```
int _struct_IO_ITEM_HEADER::level
```

5.2.2.5 `type`

```
unsigned long _struct_IO_ITEM_HEADER::type
```

5.2.2.6 `use_extension`

```
int _struct_IO_ITEM_HEADER::use_extension
```

5.2.2.7 `user_flag`

```
int _struct_IO_ITEM_HEADER::user_flag
```

5.2.2.8 version

```
unsigned _struct_IO_ITEM_HEADER::version
```

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

- [io_basic.h](#)

5.3 atmospheric_profile Struct Reference

Atmospheric profile as stored in atmprof*.dat files - the actually used columns only.

```
#include <mc_atmprof.h>
```

Data Fields

- double [aatm](#) [5]
See ATMA CORSIKA inputs card.
- double * [alt_km](#)
Altitude a.s.l.
- char * [atmprof_fname](#)
Original name of atmospheric profile loaded.
- int [atmprof_id](#)
Profile ID number ('atmprof<i>.dat') or 99.
- double [batm](#) [5]
See ATMB CORSIKA inputs card.
- double [catm](#) [5]
See ATMC CORSIKA inputs card.
- double [datm](#) [5]
Inverse of catm values (if non-zero)
- int [have_lay5_param](#)
Is 1 if the 5-layer CORSIKA built-in parametrization is known, 0 if not.
- double [hlay](#) [6]
Layer boundaries a.s.l.
- double [htoa](#)
Height (a.s.l.) at top of atmosphere [cm].
- unsigned [n_alt](#)
Number of altitude levels.
- double [obslev](#)
Observation level [cm], a.s.l., as used in CORSIKA.
- double * [refidx_m1](#)
Index of refraction minus one (n-1) at given level.
- double * [rho](#)
Density [g/cm³] at each level.
- double * [thick](#)
Vertical column density from space to given level [g/cm²].
- double [thickl](#) [6]
Atmospheric thickness at given hlay heights.

5.3.1 Detailed Description

5.3.2 Field Documentation

5.3.2.1 alt_km

```
double* atmospheric_profile::alt_km
```

[km] at each level

5.3.2.2 have_lay5_param

```
int atmospheric_profile::have_lay5_param
```

5.3.2.3 hlay

```
double atmospheric_profile::hlay[6]
```

[cm]; see ATMLAY CORSIKA inputs card

5.3.2.4 obslev

```
double atmospheric_profile::obslev
```

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

- [mc_atmprof.h](#)

5.4 bunch Struct Reference

Photons collected in bunches of identical direction, position, time, and wavelength.

```
#include <mc_tel.h>
```

Data Fields

- float [ctime](#)
Arrival time (ns)
- float [cx](#)
- float [cy](#)
Direction cosines of photon direction.
- float [lambda](#)
Wavelength in nanometers or 0.
- float [photons](#)
Number of photons in bunch.
- float [x](#)
- float [y](#)
Arrival position relative to telescope (cm)
- float [zem](#)
Height of emission point above sea level (cm)

5.4.1 Detailed Description

The wavelength will normally be unspecified as produced by CORSIKA ($\lambda=0$).

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

- [mc_tel.h](#)

5.5 camera_electronics Struct Reference

Parameters of the electronics of a telescope.

Data Fields

- int [simulated](#)
Is 1 if the signal simulation was done.
- int [telescope](#)
Telescope sequence number.

5.5.1 Field Documentation

5.5.1.1 simulated

```
int camera_electronics::simulated
```

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

- [sim_skeleton.c](#)

5.6 compact_bunch Struct Reference

The [compact_bunch](#) struct is equivalent to the bunch struct except that we try to use less memory.

```
#include <mc_tel.h>
```

Data Fields

- short [ctime](#)
 $ctime \times 10$ (0.1ns) after subtracting offset
- short [cx](#)
- short [cy](#)
 $cx, cy \times 30000$
- short [lambda](#)
(nm) or 0
- short [log_zem](#)
 $\log_{10}(zem) \times 1000$
- short [photons](#)
 $ph \times 100$
- short [x](#)
- short [y](#)
 $x, y \times 10$ (mm)

5.6.1 Detailed Description

And that has a number of limitations: 1) Bunch sizes must be less than 327. 2) photon impact points in a horizontal plane through the centre of each detector sphere must be less than 32.7 m from the detector centre in both x and y coordinates. Thus, $\sec(z) * R < 32.7$ m is required, with 'z' being the zenith angle and 'R' the radius of the detector sphere. When accounting for multiple scattering and Cherenkov emission angles, the actual limit is reached even earlier than that. 3) Only times within 3.27 microseconds from the time, when the primary particle propagated with the speed of light would cross the altitude of the sphere centre, can be treated. For large zenith angle observations this limits horizontal core distances to about 1000 m. For efficiency reasons, no checks are made on these limits.

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

- [mc_tel.h](#)

5.7 cubic_params Struct Reference

Cubic spline interpolation (natural cubic splines = scheme 3, clamped cubic splines = scheme 4)

```
#include <rpolator.h>
```

Data Fields

- double **a**
- double **b**
- double **c**
- double **d**

$$r = xp - x[i], \quad yp = a + b*r + c*r^2 + d*r^3 = ((d*r + c) * r + b) * r + a;$$

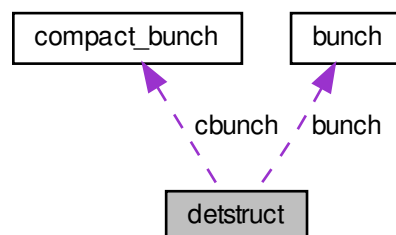
The documentation for this struct was generated from the following file:

- [rpolator.h](#)

5.8 detstruct Struct Reference

A structure describing a detector and linking its photons bunches to it.

Collaboration diagram for detstruct:



Data Fields

- int **available_bunch**
- int **bits**
- struct **bunch** * **bunch**
- struct **compact_bunch** * **cbunch**
- int **dclass**
- double **dx**
- double **dy**
- char **ext_fname** [60]
- int **external_bunches**
- int **geo_type**
- int **iarray**
- int **idet**
- int **next_bunch**
- double **photons**
- double **r**
- double **r0**
- double **sampling_area**
- int **sens_type**
- int **shrink_cycle**
- int **shrink_factor**
- double **x**
- double **x0**
- double **y**
- double **y0**
- double **z0**

5.8.1 Detailed Description

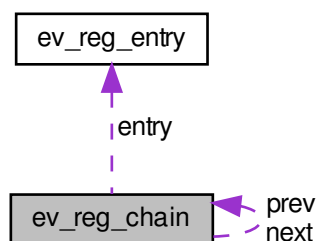
The documentation for this struct was generated from the following file:

- [iact.c](#)

5.9 ev_reg_chain Struct Reference

Use a double-linked list for the registry.

Collaboration diagram for ev_reg_chain:



Data Fields

- struct [ev_reg_entry](#) * [entry](#)
The current entry.
- struct [ev_reg_chain](#) * [next](#)
- struct [ev_reg_chain](#) * [prev](#)

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

- [eventio_registry.c](#)

5.10 ev_reg_entry Struct Reference

Data Fields

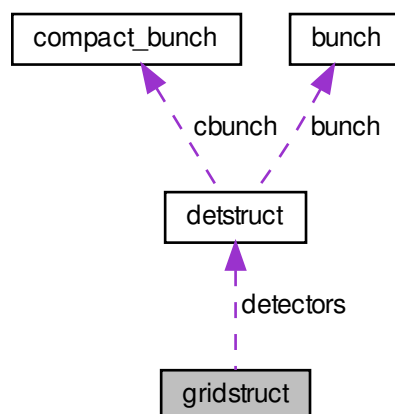
- char * [description](#)
Optional longer description of the data block.
- char * [name](#)
The data block name (short)
- unsigned long [type](#)
The data block type number.

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

- [io_basic.h](#)

5.11 gridstruct Struct Reference

Collaboration diagram for gridstruct:



Data Fields

- struct [detstruct](#) ** **detectors**
- int **idet**
- int **ndet**

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

- [iact.c](#)

5.12 incpath Struct Reference

An element in a linked list of include paths.

```
#include <fileopen.h>
```

Collaboration diagram for incpath:



Data Fields

- struct [incpath](#) * **next**
The next element.
- char * [path](#)
The path name.

5.12.1 Detailed Description

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

- [fileopen.h](#)

5.13 linked_string Struct Reference

The [linked_string](#) is mainly used to keep CORSIKA input.

```
#include <mc_tel.h>
```

Collaboration diagram for linked_string:



Data Fields

- struct [linked_string](#) * **next**
- char * **text**

5.13.1 Detailed Description

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

- [mc_tel.h](#)

5.14 mc_options Struct Reference

Options of the simulation passed through to low-level functions.

Data Fields

- int [always_with_aweight](#)
Make MCEvent output always with area weights.
- const char * [input_fname](#)
Input file name.
- long [iobuf_max](#)
Size limit for I/O buffers.

5.14.1 Detailed Description

5.14.2 Field Documentation

5.14.2.1 always_with_aweight

```
int mc_options::always_with_aweight
```

5.14.2.2 input_fname

```
const char* mc_options::input_fname
```

5.14.2.3 iobuf_max

```
long mc_options::iobuf_max
```

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

- [sim_skeleton.c](#)

5.15 mc_run Struct Reference

Basic parameters of the CORSIKA run.

Data Fields

- double [area](#)
Area over which offsets may be scattered [m^2].
- int [atmosphere](#)
Model atmosphere number.
- double [bfield_bx](#)
 x' component of B field [mT].
- double [bfield_bz](#)
 z component of B field [mT].
- double [bfield_rot](#)
rotation angle mag.
- double [bunchsize](#)
Cherenkov bunch size (nominal).
- int [corsika_version](#)
*CORSIKA version number * 1000.*
- double [e_max](#)
Upper limit of simulated energies [TeV].
- double [e_min](#)
Lower limit of simulated energies [TeV].
- double [height](#)
Height of observation level [m].
- int [high_E_detail](#)
More details on high E interaction model (version etc.)
- int [high_E_model](#)

- *High energy interaction model flags.*
- int [iact_options](#)
Option flags in CORSIKA (VOLUMEDET etc.)
- int [low_E_detail](#)
More details on low E interaction model (version etc.)
- int [low_E_model](#)
Low energy interaction model flags.
- int [num_arrays](#)
Number of arrays simulated.
- int [num_showers](#)
Number of showers simulated in run.
- double [phi_max](#)
Upper limit of azimuth angle [degrees].
- double [phi_min](#)
Lower limit of azimuth angle [degrees].
- double [radius](#)
Radius within which cores are thrown at random.
- double [radius1](#)
Distance parameter 1 in CSCAT [m].
- double [radius2](#)
Distance parameter 2 in CSCAT [m].
- int [run](#)
Run number.
- double [slope](#)
Spectral index of power-law spectrum.
- double [start_depth](#)
Atmospheric depth where primary particle started.
- double [theta_max](#)
Upper limit of zenith angle [degrees].
- double [theta_min](#)
Lower limit of zenith angle [degrees].
- double [viewcone_max](#)
Maximum of VIEWCONE range [degrees].
- double [viewcone_min](#)
Minimum of VIEWCONE range [degrees].
- double [wlen_max](#)
Upper limit of Cherenkov wavelength range [nm].
- double [wlen_min](#)
Lower limit of Cherenkov wavelength range [nm].

5.15.1 Field Documentation

5.15.1.1 area

```
double mc_run::area
```

5.15.1.2 atmosphere

```
int mc_run::atmosphere
```

5.15.1.3 bfield_bx

```
double mc_run::bfield_bx
```

5.15.1.4 bfield_bz

```
double mc_run::bfield_bz
```

5.15.1.5 bfield_rot

```
double mc_run::bfield_rot
```

N -> geogr. N [degrees].

5.15.1.6 bunchsize

```
double mc_run::bunchsize
```

5.15.1.7 corsika_version

```
int mc_run::corsika_version
```

5.15.1.8 high_E_model

```
int mc_run::high_E_model
```

5.15.1.9 low_E_model

```
int mc_run::low_E_model
```

5.15.1.10 num_arrays

```
int mc_run::num_arrays
```

5.15.1.11 num_showers

```
int mc_run::num_showers
```

5.15.1.12 radius

```
double mc_run::radius
```

[m]

5.15.1.13 run

```
int mc_run::run
```

5.15.1.14 start_depth

```
double mc_run::start_depth
```

5.15.1.15 viewcone_max

```
double mc_run::viewcone_max
```

5.15.1.16 viewcone_min

```
double mc_run::viewcone_min
```

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

- [sim_skeleton.c](#)

5.16 photo_electron Struct Reference

A photo-electron produced by a photon hitting a pixel.

```
#include <mc_tel.h>
```

Data Fields

- double [atime](#)
The time [ns] when the photon hit the pixel.
- int [lambda](#)
The wavelength of the photon.
- int [pixel](#)
The pixel that was hit.

5.16.1 Detailed Description

5.16.2 Field Documentation

5.16.2.1 `atime`

```
double photo_electron::atime
```

5.16.2.2 `lambda`

```
int photo_electron::lambda
```

5.16.2.3 `pixel`

```
int photo_electron::pixel
```

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

- [mc_tel.h](#)

5.17 `photon_bunch` Struct Reference

Data Fields

- double **ctime**
- double **dt**
- double **dx**
- double **dy**
- double **lambda**
- double **photons**
- double **tol**
- double **u**
- double **v**
- double **w**
- double **weight**
- double **xem**
- double **xol**
- double **yem**
- double **yol**
- double **zem**

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

- [fake_corsika.c](#)

5.18 pm_camera Struct Reference

Parameters of a telescope camera (pixels, ...)

Data Fields

- int [telescope](#)
Telescope sequence number.

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

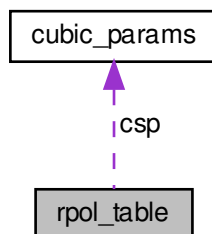
- [sim_skeleton.c](#)

5.19 rpol_table Struct Reference

Structure describing an interpolation table, interpolation scheme and selected options.

```
#include <rpolator.h>
```

Collaboration diagram for rpol_table:



Data Fields

- double [aux](#)
This auxiliary value may be useful to the application but is not used internally.
- int [clipping](#)
0: Extrapolate with edge value, 1: zero outside range.
- [CsplinePar](#) * [csp](#)
Cubic spline parameters (scheme 3 and 4 only), need one-time initialisation.
- double [dx](#)
- double [dxi](#)
Step size in x and inverse of it, for equidistant only.
- double [dy](#)
- double [dyi](#)

- Step size in y and inverse of it, for equidistant only (2-D).*

 - int **equidistant**

Equidistant support points make life much easier (bit 0: x, bit 1: y).
- char * **fname**

Name of the file from which the table was loaded (includes all options).
- int **logs**
- int **ndim**

1 or 2 dimension(s), for the independent variable(s) that is.
- size_t **nx**
- size_t **ny**

No.
- char * **options**

Options used in option parameter or NULL pointer.
- int **remapped**

If user code remaps x, y, and/or z values w.r.t.
- int **scheme**

Requested interpolation scheme (0: nearest, 1: linear, 2: quadratic, 3: natural cubic spline, 4: clamped cubic spline).
- int **use_count**

Indicates how often a table is in use, but not safe enough to make it a smart pointer.
- double * **x**

Supporting points in x.
- int **xlog**
- double **xmax**

Range covered in x.
- double **xmin**
- double **xrise**

+1 if x values are in ascending order; -1 for descending
- double * **y**

Supporting points in y (if ndim=2 and ny>1)
- int **ylog**
- double **ymax**

Range covered in y (if ndim=2)
- double **ymin**
- double **yrise**

+1 if y values are in ascending order; -1 for descending
- double * **z**

*Table values (size nx for 1-dim or nx*ny for 2-dim)*
- int **zlog**

Log applied to any x/y axis, to x axis, y axis, z axis?
- double **zmax**

Range covered in result values.
- double **zmin**
- double * **zxmax**

Optional max.
- double * **zxmin**

Optional min.
- int **zxreq**

Flag activated when options indicate that a set of zxmax values should be provided.

5.19.1 Detailed Description

5.19.2 Field Documentation

5.19.2.1 aux

double rpol_table::aux

5.19.2.2 clipping

int rpol_table::clipping

5.19.2.3 csp

CsplinePar* rpol_table::csp

5.19.2.4 dxi

double rpol_table::dxi

5.19.2.5 dyi

double rpol_table::dyi

5.19.2.6 equidistant

int rpol_table::equidistant

5.19.2.7 fname

char* rpol_table::fname

5.19.2.8 ndim

int rpol_table::ndim

5.19.2.9 ny

```
size_t rpol_table::ny
```

of entries in x and (optional) y points.

5.19.2.10 options

```
char* rpol_table::options
```

5.19.2.11 remapped

```
int rpol_table::remapped
```

the table input, it should mark this here to avoid repeating it. (Make sure to call `rpol_check_equi_range()` after you remap!)

5.19.2.12 scheme

```
int rpol_table::scheme
```

5.19.2.13 use_count

```
int rpol_table::use_count
```

5.19.2.14 zxmax

```
double* rpol_table::zxmax
```

of z values along each y line (at each x value)

5.19.2.15 zxmin

```
double* rpol_table::zxmin
```

of z values along each y line (at each x value)

5.19.2.16 zxreq

```
int rpol_table::zxreq
```

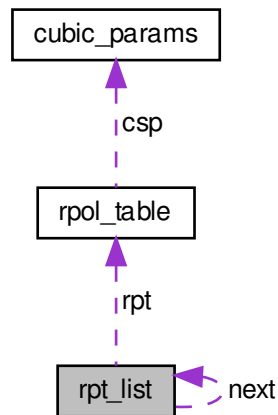
The documentation for this struct was generated from the following file:

- [rpolator.h](#)

5.20 rpt_list Struct Reference

Registered interpolation tables allow re-use of tables without having to load them again.

Collaboration diagram for rpt_list:



Data Fields

- struct `rpt_list` * `next`
- struct `rpol_table` * `rpt`

5.20.1 Detailed Description

Available for simple (2-column) 1-D interpolation and for rectangular-grid 2-D interpolation with y grid values in header line, x grid values in first column.

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

- [rpolator.c](#)

5.21 shower_extra_parameters Struct Reference

Extra shower parameters of unspecified nature.

```
#include <mc_tel.h>
```

Data Fields

- float * [fparam](#)
Space for extra floats, at least of size nparam.
- long [id](#)
May identify to the user what the parameters should mean.
- int * [iparam](#)
Space for extra integer parameters, at least of size nparam.
- int [is_set](#)
May be reset after writing the parameter block and must thus be set to 1 for each shower for which the extra parameters should get recorded.
- size_t [nparam](#)
Number of extra floating-point parameters.
- size_t [niparam](#)
Number of extra integer parameters.
- double [weight](#)
To be used if the weight of a shower may change during processing, e.g.

5.21.1 Detailed Description

Useful for things to be used like in the event header but which may only become available while processing a shower. Should be initialized with the `init_shower_extra_parameters(int ni_max, int nf_max)` function.

5.21.2 Field Documentation

5.21.2.1 fparam

```
float* shower_extra_parameters::fparam
```

5.21.2.2 id

```
long shower_extra_parameters::id
```

5.21.2.3 iparam

```
int* shower_extra_parameters::iparam
```

5.21.2.4 is_set

```
int shower_extra_parameters::is_set
```

5.21.2.5 nfparam

```
size_t shower_extra_parameters::nfparam
```

5.21.2.6 niparam

```
size_t shower_extra_parameters::niparam
```

5.21.2.7 weight

```
double shower_extra_parameters::weight
```

when shower processing can be aborted depending on how quickly the electromagnetic component builds up and the remaining showers may have a larger weight to compensate for that. For backwards compatibility this should be set to 1.0 when no additional weight is needed.

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

- [mc_tel.h](#)

5.22 simulated_shower_parameters Struct Reference

Basic parameters of a simulated shower.

Data Fields

- double [altitude](#)
Shower direction altitude above horizon.
- int [array](#)
Array number = shower usage number.
- double [aweight](#)
Area weight to be used with non-uniform.
- double [azimuth](#)
Shower direction azimuth [deg].
- double [clongi](#) [1071]
Vertical profile of Cherenkov light emission.
- double [cmax](#)
*Depth of maximum of Cherenkov light emission [g/cm**2].*
- double [core_dist_3d](#)
Distance of core from reference point.
- double [elongi](#) [1071]
Vertical profile of all electrons+positrons.
- double [emax](#)
Depth of shower maximum from positrons and electrons.
- double [energy](#)
Shower energy [TeV].

- double `h1int`
Height a.s.l.
- int `have_longi`
Indicates if vertical profiles were found in data.
- double `hmax`
Height of shower maximum (from x_{\max} above) [m] a.s.l.
- int `particle`
Primary particle type [CORSIKA code].
- int `shower`
Shower number.
- double `step_longi`
*Step size of vertical profiles [g/cm**2].*
- double `tel_core_dist_3d` [MAX_TEL]
Offset of telescopes from shower axis.
- double `x0`
Atmospheric depth where particle was started [g/cm²].
- double `xcore`
- double `xlongi` [1071]
Vertical profile of all particles.
- double `xmax`
*Depth of shower maximum from all particles [g/cm**2].*
- double `ycore`
- double `zcore`
Shower core position [m].

5.22.1 Field Documentation

5.22.1.1 `aweight`

`double simulated_shower_parameters::aweight`

core-position sampling [m²].

5.22.1.2 `clongi`

`double simulated_shower_parameters::clongi[1071]`

5.22.1.3 `elongi`

`double simulated_shower_parameters::elongi[1071]`

5.22.1.4 `h1int`

`double simulated_shower_parameters::h1int`

of first interaction [m].

5.22.1.5 have_longi

```
int simulated_shower_parameters::have_longi
```

If not, the followig numbers should be all zeroes.

5.22.1.6 hmax

```
double simulated_shower_parameters::hmax
```

5.22.1.7 step_longi

```
double simulated_shower_parameters::step_longi
```

5.22.1.8 x0

```
double simulated_shower_parameters::x0
```

5.22.1.9 xlongi

```
double simulated_shower_parameters::xlongi[1071]
```

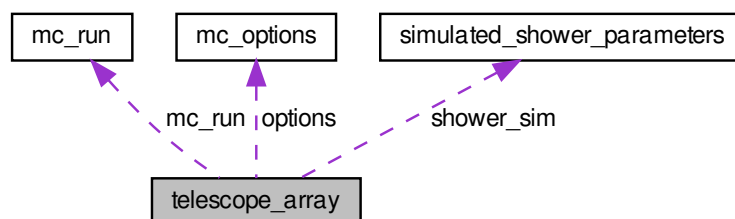
The documentation for this struct was generated from the following file:

- [sim_skeleton.c](#)

5.23 telescope_array Struct Reference

Description of telescope position, array offets and shower parameters.

Collaboration diagram for telescope_array:



Data Fields

- double [altitude](#)
Nominal altitude angle of telescope system [deg].
- double [aweight](#) [MAX_ARRAY]
Area weight for non-uniformly distributed core offsets.
- double [azimuth](#)
Nominal azimuth angle of telescope system [deg].
- int **event**
- int [max_tel](#)
Maximum number of telescopes acceptable (MAX_TEL)
- struct [mc_run](#) **mc_run**
- int [narray](#)
Number of arrays with random shifts per shower.
- int [ntel](#)
Number of telescopes simulated per array.
- double [obs_height](#)
Height of observation level [cm].
- struct [mc_options](#) **options**
File names etc.
- double [refpos](#) [3]
Reference position with respect to obs.
- double [rtel](#) [MAX_TEL]
Radius of spheres enclosing telescopes [cm].
- int **run**
- struct [simulated_shower_parameters](#) **shower_sim**
- double [source_altitude](#)
Altitude of assumed source.
- double [source_azimuth](#)
Azimuth of assumed source.
- double [toff](#)
Time offset from first interaction to the moment.
- int [with_aweight](#)
Is 1 if input data came with weights.
- double [xoff](#) [MAX_ARRAY]
X offsets of the randomly shifted arrays [cm].
- double [xtel](#) [MAX_TEL]
X positions of telescopes ([cm] -> north)
- double [yoff](#) [MAX_ARRAY]
Y offsets of the randomly shifted arrays [cm].
- double [ytel](#) [MAX_TEL]
Y positions of telescopes ([cm] -> west)
- double [ztel](#) [MAX_TEL]
Z positions of telescopes ([cm] -> up)

5.23.1 Detailed Description

5.23.2 Field Documentation

5.23.2.1 altitude

```
double telescope_array::altitude
```

5.23.2.2 aweight

```
double telescope_array::aweight [MAX_ARRAY]
```

(may be 0 when older data with uniform offsets are read). [cm²]

5.23.2.3 azimuth

```
double telescope_array::azimuth
```

5.23.2.4 options

```
struct mc_options telescope_array::options
```

5.23.2.5 refpos

```
double telescope_array::refpos[3]
```

level [cm]

5.23.2.6 source_altitude

```
double telescope_array::source_altitude
```

5.23.2.7 source_azimuth

```
double telescope_array::source_azimuth
```

5.23.2.8 toff

```
double telescope_array::toff
```

when the extrapolated primary flying with the vacuum speed of light would be at the observation level.

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

- [sim_skeleton.c](#)

5.24 telescope_optics Struct Reference

Parameters describing the telescope optics.

Data Fields

- [int telescope](#)
Telescope sequence number.

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

- [sim_skeleton.c](#)

5.25 telpos_struct Struct Reference

Data Fields

- [cors_dbl_t r](#)
- [cors_dbl_t x](#)
- [cors_dbl_t y](#)
- [cors_dbl_t z](#)

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

- [fake_corsika.c](#)

5.26 test_struct Struct Reference

Data Fields

- [uint8_t bvar](#) [2]
- [uint16_t cnt16var](#) [4]
- [uint32_t cnt32var](#) [6]
- [size_t cntvar](#) [8]
- [size_t cntzvar](#) [6]
- [double dvar](#) [2]
- [double fvar](#) [2]
- [double hvar](#) [2]
- [int16_t i16var](#) [2]
- [int32_t i32var](#) [2]
- [int8_t i8var](#) [2]
- [int ilvar](#) [2]
- [int isvar](#) [2]
- [long lvar](#) [2]
- [size_t nbvar](#)
- [int16_t scnt16var](#) [10]
- [int32_t scnt32var](#) [12]
- [ssize_t scntvar](#) [14]
- [ssize_t scntzvar](#) [12]
- [char str16var](#) [10]
- [char str32var](#) [10]
- [char strvvar](#) [10]
- [short svar](#) [3]
- [uint16_t u16var](#) [2]
- [uint32_t u32var](#) [2]
- [uint8_t u8var](#) [2]

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

- [testio.c](#)

5.27 warn_specific_data Struct Reference

A struct used to store thread-specific data.

Data Fields

- char *****(*** aux_function**)(void)
- int **buffered**
- void(*** log_function**)(const char *, const char *, int, int)
- FILE *** logfile**
- const char ***** [logfname](#)
The name of the log file.
- char **output_buffer** [2048]
- void(*** output_function**)(const char *)
- int **recursive**
- char **saved_logfname** [256]
- int **warninglevel**
- int **warningmode**

5.27.1 Detailed Description

5.27.2 Field Documentation

5.27.2.1 logfname

```
const char* warn_specific_data::logfname
```

Used only when opening the file.

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

- [warning.c](#)

6 File Documentation

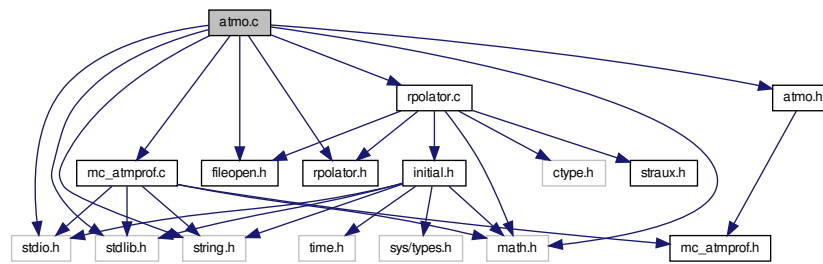
6.1 atmo.c File Reference

Use of tabulated atmospheric profiles and atmospheric refraction.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "atmo.h"
#include "fileopen.h"
#include "rpolator.h"
#include "rpolator.c"
```

```
#include "mc_atmprof.c"
```

Include dependency graph for atm.c:



Macros

- `#define NLAYX 11`
- `#define M_PI (3.14159265358979323846264338327950288)`
/
- `#define WITH_RPOLATOR_CSPLINE 1` /* Take advantage of cubic splines if we have them available */
- `#define FAST_INTERPOLATION always`
- `#define FAST_INTERPOLATION2 always`
- `#define FAST_INTERPOLATION3 always`
- `#define WITH_THICKX_DIRECT`
- `#define UNUSED(x) UNUSED_ ## x`
- `#define MAX_PROFILE 120`
- `#define MAX_FAST_PROFILE 40000`
- `#define _XSTR(s) _STR(s)`
Expand a macro first and then enclose in string.
- `#define _STR(s) #s`
Enclose in string without macro expansion.
- `#define SHOW_MACRO(s)`
The following shows the same output after a "`#define XYZ`" and a "`#define XYZ 1`" (or compiled with "`-DXYZ`")
- `int atmosphere`
The atmospheric profile number, 0 for built-in.
- `static char * atmprof_name = NULL`
File name for atmospheric profile table.
- `static int num_prof`
Number of levels in original table.
- `static double p_alt [MAX_PROFILE]`
Altitude levels in given table (converted to [cm], upward order).
- `static double p_alt_rev [MAX_PROFILE]`
Altitude levels in given table (converted to [cm], reversed order).
- `static double p_rho [MAX_PROFILE]`
Density at given levels [g/cm³] (upward height order)
- `static double p_rho_rev [MAX_PROFILE]`
Density at given levels [g/cm³], reversed order.
- `static double p_thick [MAX_PROFILE]`
Atmospheric thickness at given levels [g/cm²].

- static double `p_log_rho` [MAX_PROFILE]
Log of density at given levels (for better interpolation)
- static double `p_log_thick` [MAX_PROFILE]
Log of atmospheric thickness at given levels (upward height order)
- static double `p_log_thick_rev` [MAX_PROFILE]
Log of atmospheric thickness at given levels (reversed order)
- static double `p_log_n1` [MAX_PROFILE]
Log of (index of refraction minus 1.0) at given levels.
- static double `p_bend_ray_hori_a` [MAX_PROFILE]
Coefficient for horizontal displacement refraction effect (reversed order)
- static double `p_bend_ray_time_a` [MAX_PROFILE]
Coefficient for arrival time effect by refraction, density dependence (reversed order)
- static double `p_bend_ray_time0` [MAX_PROFILE]
Coefficient for arrival time effect by refraction, altitude dependence.
- static `CsplinePar` * `cs_alt_log_rho`
Cubic spline parameters for log(density) versus altitude interpolation.
- static `CsplinePar` * `cs_alt_log_thick`
Cubic spline parameters for log(thickness) versus altitude interpolation.
- static `CsplinePar` * `cs_alt_log_n1`
Cubic spline parameters for log(n-1) versus altitude interpolation.
- static `CsplinePar` * `cs_log_thick_alt`
Cubic spline parameters for altitude versus log(thickness) reverse interpolation.
- static `CsplinePar` * `cs_bend_rho_hori_a`
Cubic spline parameters for horizontal displacement refraction effect.
- static `CsplinePar` * `cs_bend_rho_time_a`
Cubic spline parameters for arrival time effect by refraction, density dependence.
- static `CsplinePar` * `cs_bend_alt_time0`
Cubic spline parameters for arrival time effect by refraction, altitude dependence.
- static double `top_of_atmosphere` = 112.83e5
Height of top of atmosphere [cm], = `p_alt[num_prof-1]`.
- static double `top_of_atm_table`
The heighest entry in the atmospheric profile table.
- static double `bottom_of_atmosphere` = 0.
Bottom is normally at sea level but table could differ, = `p_alt[0]`.
- static double `bottom_log_thickness`
Thickness at bottom, depending on profile, = `p_log_thick[0]`.
- static double `top_log_thickness`
Thickness at top is zero, thus using an extrapolation from the second last value.
- static double `top_layer_2nd_altitude`
Second highest altitude tabulated, using exponential density profile above that.
- static double `top_layer_rho0`
Sea-level altitude matching profile in top layer, without scaling for thickness.
- static double `top_layer_cfacs`
Scaling factor needed to match thickness at second last level.
- static double `top_layer_hscale`
Height scale of exponention density profile in top layer.
- static double `top_layer_hscale_rho0_cfacs`
*`top_layer_rho0` * `top_layer_cfacs` * `top_layer_hscale`*
- static double `top_layer_hscale_rho0_cfacs_inv`
1.
- static double `top_layer_exp_top`

- exp(-top_of_atmosphere/top_layer_hscale)*
- static double * [fast_p_alt](#)
Equidistant steps in altitude.
- static double * [fast_p_log_rho](#)
Log(rho) at fast_p_alt steps.
- static double * [fast_p_log_thick](#)
Log(thick) at fast_p_alt steps.
- static double * [fast_p_log_n1](#)
Log(n-1) at fast_p_alt steps.
- static double [fast_h_fac](#)
Inverse of height difference per step.
- static double * [fast_p_thick](#)
Direct interpolation of thickness at fast_p_alt steps.
- static double * [fast_p_rho](#)
Direct interpolation of density at fast_p_alt steps.
- static double * [fast_p_n1](#)
Direct interpolation of n-1 at fast_p_alt steps.
- static double * [fast_p_eq_log_thick](#)
Equidistant steps in log(thick)
- static double * [fast_p_heigh_rev](#)
Altitude at given log(thick) steps.
- static double [fast_log_thick_fac](#)
Inverse of log(thickness) difference per step.
- static double * [fast_p_bend_ray_hori_a](#)
Coefficient for horizontal displacement refraction effect.
- static double * [fast_p_bend_ray_time_a](#)
Coefficient for arrival time effect by refraction, density dependence.
- static double * [fast_p_bend_ray_time0](#)
Coefficient for arrival time effect by refraction, altitude dependence.
- static double * [fast_p_rho_rev](#)
Density steps used in fast interpolation, in order of incr.
- static double **fast_rho_fac**
- static double **etadsn** = 0.000283 * 994186.38 / 1222.656
- static double [observation_level](#)
Altitude [cm] of observation level.
- static double **obs_level_refidx**
- static double **obs_level_thick**
- static void [init_refraction_tables](#) ()
Initialize tables needed for atmospheric refraction.
- static void [init_fast_interpolation](#) ()
An alternate interpolation method (which requires that the table is sufficiently fine-grained and equidistant) has to be initialized first.
- static void [init_corsika_atmosphere](#) ()
Take the atmospheric profile from CORSIKA built-in functions.
- static void [init_atmosphere_from_text_file](#) ()
Initialize atmospheric profiles.
- static void **init_atmosphere_from_atmprof** (void)
- static double **sum_log_dev_sq** (double a, double b, double c, int np, double *h, double *t, double *rho)
- static double **atm_exp_fit** (double h1, double h2, double *ap, double *bp, double *cp, double *s0, int *npp)
Fit one atmosphere layer by an exponential density model.
- void **free_atmo_csplines** (void)

- static void `trace_ray_planar` (double emlev, double olev, double za, double step, double *xo, double *to, double *so)
Trace a downward light ray in a planar atmosphere.
- static void `init_common_atmosphere` (void)
Second-stage part of atmospheric profile initialization is common to CORSIKA default profile and tabulated profiles.
- void `atmset_` (int *iatmo, double *obslev)
Set number of atmospheric model profile to be used.
- void `atmnam_` (const char *aname, double *obslev)
Instead of setting the atmospheric profile by number, it gets set by name, also indicated by setting profile number to 99.
- void `atm_init` (AtmProf *aprof)
This variant is not usable from the FORTRAN code side, thus no underscore at the end of the function name.
- double `rhofx_` (double *height)
Density of the atmosphere as a function of altitude.
- double `thickx_` (double *height)
*Atmospheric thickness [g/cm**2] as a function of altitude.*
- double `refim1x_` (double *height)
Index of refraction minus 1 as a function of altitude [cm].
- double `refidx_` (double *height)
Index of refraction as a function of altitude [cm].
- double `heighx_` (double *thick)
*Altitude [cm] as a function of atmospheric thickness [g/cm**2].*
- void `raybnd_` (double *zem, `cors_dbl_t` *u, `cors_dbl_t` *v, double *w, `cors_dbl_t` *dx, `cors_dbl_t` *dy, `cors_dbl_t` *dt)
Calculate the bending of light due to atmospheric refraction.
- static double `sum_log_dev_sq` (double a, double b, double c, int np, double *h, double *t, double *UNUSED←ED(rho))
Measure of deviation of model layers from tables.
- static double `fn_thick` (double h, int nl, double *hl, double *a, double *b, double *c)
Corresponding to CORSIKA built-in function THICK; only used to show fit results.
- static double `fn_rhof` (double h, int nl, double *hl, double *UNUSED(a), double *b, double *c)
Corresponding to CORSIKA built-in function RHOF; only used to show fit results.
- void `atmfit_` (int *nlp, double *hlay, double *aatm, double *batm, double *catm)
Fit the tabulated density profile for CORSIKA EGS part.
- void `show_atmo_macros` (void)

6.1.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2019/09/03 14:54:51 \$

Version

CVS \$Revision: 1.42 \$

This file provides code for use of external atmospheric models (in the form of text-format tables) with the CORSIKA program. Six atmospheric models as implemented in the MODTRAN program and as tabulated in MODTRAN documentation (F.X. Kneizys et al. 1996, 'The MODTRAN 2/3 Report and LOWTRAN 7 Model', Phillips Laboratory, Hanscom AFB, MA 01731-3010, U.S.A.) are provided as separate files (atmprof1.dat ... atmprof6.dat). User-provided atmospheric models should be given model numbers above 6. For model number 99 you may actually define a non-standard file name, rather than atmprof99.dat.

Note that for the Cherenkov part and the hadronic (and muon) part of CORSIKA the table values are directly interpolated but the electron/positron/gamma part (derived from EGS) uses special layers (at present 4 with exponential density decrease and the most upper layer with constant density). Parameters of these layers are fitted to tabulated values but not every possible atmospheric model fits very well with an exponential profile. You are advised to check that the fit matches tabulated values to sufficient precision in the altitude ranges of interest to you. Try to adjust layer boundary altitudes in case of problems. The propagation of light without refraction (as implemented in CORSIKA, unless using the CURVED option) and with refraction (as implemented by this software) assumes a plane-parallel atmosphere. Instead of a single set of starting layer boundaries, newer version (Jan. 2019) try different sets of boundaries and use the most promising set for further fit improvement.

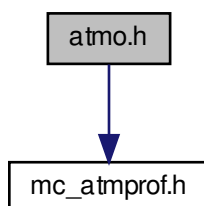
There are different ways to load and interpolate an atmospheric profile table. The usual procedure is that after loading the values from the file, the nearly exponential structure is taken advantage of and a relatively large number (order 10k) of equidistant support points are pre-interpolated from the original non-equidistant levels of the input point, saving binary-search of the matching interval for each interpolation. This FAST_INTERPOLATION option is activated by default but can be disabled by compiling with '-DNO_FAST_INTERPOLATION' (or '-DNO_FAST_INTERPOLATION2' if only fast interpolation for the reverse direction, from thickness to altitude, should be disabled.) The initial pre-interpolation used to be a linear interpolation in altitude versus log(density) or log(thickness) or log(n-1), respectively but with the 'rpolator' table interpolation code this can be turned into cubic spline interpolation. Unless problems with the CORSIKA build procedure prevent that the rpolator code and its cubic spline interpolation are intended to be the default scheme from now on (Jan. 2019). In either case for the default setting, the rpolator code can be enforced with '-DWITH_RPOLATOR' or disabled with '-DNO_RPOLATOR'. Cubic spline interpolation can be disabled with '-DNO_RPOLATOR_CSPLINE'. The fine-grained interpolation under the FAST_INTERPOLATION scheme continues to be linear in altitude versus log(density) etc.

Because the CORSIKA build procedure does not know yet about the additional source file, it gets included when compiling [atmo.c](#), unless compiled with '-DWITH_RPOLATOR_SEPARATE'.

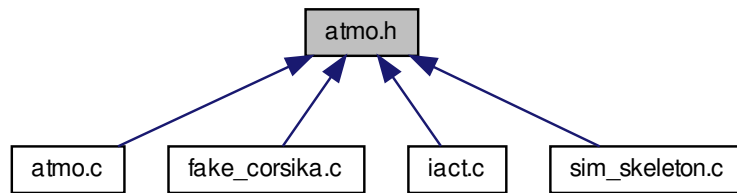
6.2 atmo.h File Reference

Use of tabulated atmospheric profiles and atmospheric refraction.

```
#include "mc_atmprof.h"
Include dependency graph for atmo.h:
```



This graph shows which files directly or indirectly include this file:



- `#define` `CORSIKA_VERSION` 6900
/
- `typedef` `double` `cors_dbl_t`
- `void` `atmset_` (`int` *iatmo, `double` *obslev)
Set number of atmospheric model profile to be used.
- `void` `atmnam_` (`const char` *aname, `double` *obslev)
Instead of setting the atmospheric profile by number, it gets set by name, also indicated by setting profile number to 99.
- `void` `atm_init` (`AtmProf` *aprof)
This variant is not usable from the FORTRAN code side, thus no underscore at the end of the function name.
- `double` `rhofx_` (`double` *height)
Density of the atmosphere as a function of altitude.
- `double` `thickx_` (`double` *height)
*Atmospheric thickness [g/cm**2] as a function of altitude.*
- `double` `refim1x_` (`double` *height)
Index of refraction minus 1 as a function of altitude [cm].
- `double` `refidx_` (`double` *height)
Index of refraction as a function of altitude [cm].
- `double` `heighx_` (`double` *thick)
*Altitude [cm] as a function of atmospheric thickness [g/cm**2].*
- `void` `raybnd_` (`double` *zem, `cors_dbl_t` *u, `cors_dbl_t` *v, `double` *w, `cors_dbl_t` *dx, `cors_dbl_t` *dy, `cors_dbl_t` *dt)
Calculate the bending of light due to atmospheric refraction.
- `void` `atmfit_` (`int` *nlp, `double` *hlay, `double` *aatm, `double` *batm, `double` *catm)
Fit the tabulated density profile for CORSIKA EGS part.
- `double` `rhof_` (`double` *height)
The CORSIKA built-in density lookup function.
- `double` `thick_` (`double` *height)
The CORSIKA built-in function for vertical atmospheric thickness (overburden).
- `double` `heigh_` (`double` *thick)
The CORSIKA built-in function for the height as a function of overburden.

6.2.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2019/07/23 16:54:21 \$

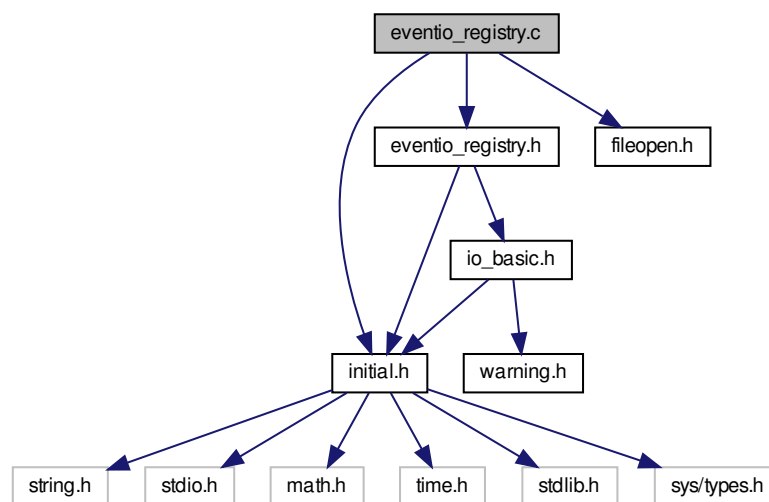
Version

CVS \$Revision: 1.12 \$

6.3 eventio_registry.c File Reference

Register and enquire about well-known I/O block types.

```
#include "initial.h"
#include "eventio_registry.h"
#include "fileopen.h"
Include dependency graph for eventio_registry.c:
```



Data Structures

- struct [ev_reg_chain](#)

Use a double-linked list for the registry.

Functions

- struct [ev_reg_entry](#) * [find_ev_reg_std](#) (unsigned long t)
Find an entry for a given type number in the registry.
- struct [ev_reg_entry](#) * [new_reg_entry](#) (unsigned long t, const char *n, const char *d)
Allocate a new entry for the registry.
- static void [read_default_registry](#) (void)
By default the registry contents will be searched in a few places.
- int [read_eventio_registry](#) (const char *fname)
Read the type names and descriptions into the registry.
- void [set_ev_reg_std](#) ()
Set the default registry search function.

Variables

- static struct [ev_reg_chain](#) * [ev_reg_start](#) = NULL

6.3.1 Detailed Description

Author

Konrad Bernloehr

Date

2014
CVS

Date

2018/09/19 12:11:36

Version

CVS

Revision

1.5

6.3.2 Function Documentation

6.3.2.1 find_ev_reg_std()

```
struct ev_reg_entry* find_ev_reg_std (
    unsigned long t )
```

This is the standard implementation being used by default where available. Here is the caller graph for this function:



6.3.2.2 read_eventio_registry()

```
int read_eventio_registry (
    const char * fname )
```

Note: this will only be done once. Here is the caller graph for this function:



6.3.2.3 set_ev_reg_std()

```
void set_ev_reg_std (
    void )
```

At least with GCC we can do this without explicitly calling it.

References `find_ev_reg_std()`, and `set_eventio_registry_hook()`.

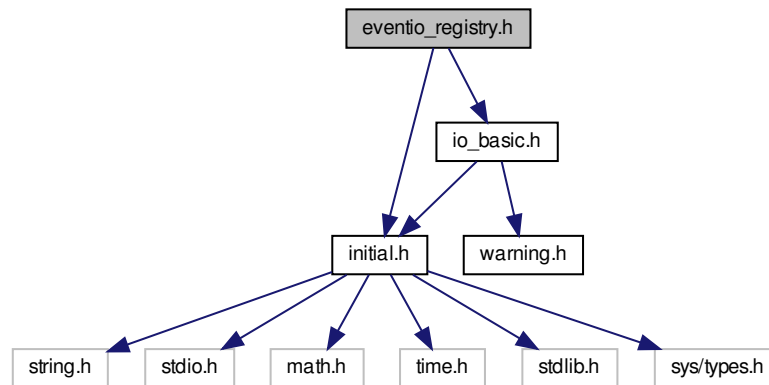
6.4 eventio_registry.h File Reference

Register and enquire about well-known I/O block types.

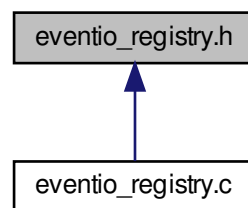
```
#include "initial.h"
```

```
#include "io_basic.h"
```

Include dependency graph for eventio_registry.h:



This graph shows which files directly or indirectly include this file:



Functions

- struct [ev_reg_entry](#) * [find_ev_reg_std](#) (unsigned long t)
Find an entry for a given type number in the registry.
- int [read_eventio_registry](#) (const char *fname)
Read the type names and descriptions into the registry.
- void [set_ev_reg_std](#) (void)
Set the default registry search function.

6.4.1 Detailed Description

Author

Konrad Bernloehr

Date

2014
CVS

Date

2014/06/01 11:33:04

Version

CVS

Revision

1.2

6.4.2 Function Documentation

6.4.2.1 find_ev_reg_std()

```
struct ev_reg_entry* find_ev_reg_std (  
    unsigned long t )
```

This is the standard implementation being used by default where available. Here is the caller graph for this function:



6.4.2.2 read_eventio_registry()

```
int read_eventio_registry (
    const char * fname )
```

Note: this will only be done once. Here is the caller graph for this function:



6.4.2.3 set_ev_reg_std()

```
void set_ev_reg_std (
    void )
```

At least with GCC we can do this without explicitly calling it.

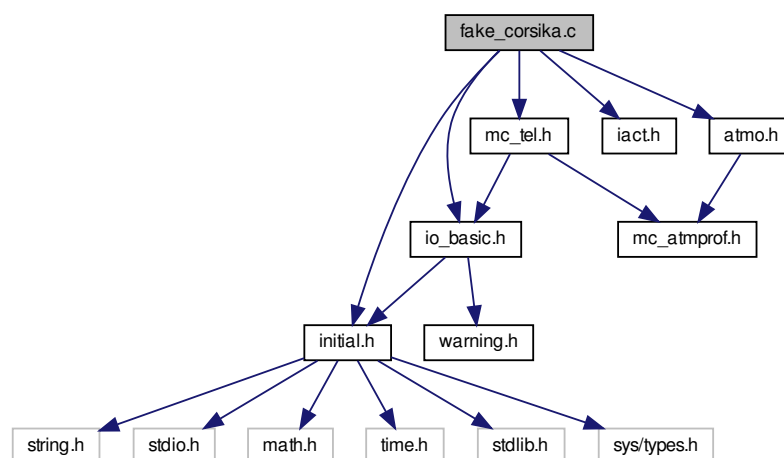
References find_ev_reg_std(), and set_eventio_registry_hook().

6.5 fake_corsika.c File Reference

Simple demonstration of how to write photon bunches without CORSIKA.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "iact.h"
#include "atmo.h"
```

Include dependency graph for fake_corsika.c:



Data Structures

- struct `photon_bunch`
- struct `telpos_struct`

Typedefs

- typedef struct `photon_bunch` **Bunch**
- typedef struct `telpos_struct` **TelPos**

Functions

- void **addhist** (const char *txt)
- double `heigh_` (double *thick)
The CORSIKA built-in function for the height as a function of overburden.
- int **main** ()
- double `rhof_` (double *height)
The CORSIKA built-in density lookup function.
- void **rmmard_** (double *a, int *b, int *c)
- double `thick_` (double *height)
The CORSIKA built-in function for vertical atmospheric thickness (overburden).

Variables

- static **Bunch** `bunch`
- `cors_dbl_t cscat1` = 0.
- `cors_dbl_t cscat2` = 0.
- static `cors_real_t evte` [273]
- static `cors_real_t evth` [273]
- char **fname** [1024]
- int **iatmo** = 1
- int **ndim** =15
- int **nprof** =9
- static int **nscat** = 1
- int **nshow** = 0
- int **nthick** =10
- double **obslev** = 2000e2
- static `cors_dbl_t prmpar` [`PRMPAR_SIZE`]
- double **profile** [9][15]
- int **ptype** =1
- double **rtel** = 10e2
- static `cors_real_t rune` [273]
- static `cors_real_t runh` [273]
- static **TelPos** `telpos`
- double **thickstep** =100

6.5.1 Detailed Description

The resulting data file will be in the same format as CORSIKA IACT output and can thus be processed with `sim_telarray` or any other program handling the same type of data.

For a much more easy-to-use tool see `lactLightEmission` in the `sim_telarray` package.

Author

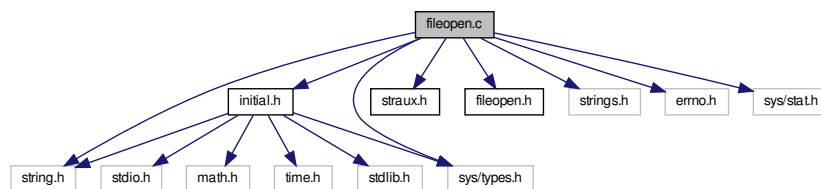
Konrad Bernloehr

6.6 fileopen.c File Reference

Allow searching of files in declared include paths (fopen replacement).

```
#include "initial.h"
#include "straux.h"
#include "fileopen.h"
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
```

Include dependency graph for `fileopen.c`:



Functions

- void `addexepath` (const char *name)
Add a path to the list of execution paths, if not already there.
- void `addpath` (const char *name)
Add a path to the list of include paths, if not already there.
- static FILE * `cmp_popen` (const char *fname, const char *mode, int compression)
Helper function for opening a compressed file through a fifo.
- void `disable_permissive_pipes` ()
Disable the permissive execution of pipes.
- void `enable_permissive_pipes` ()
Enable the permissive execution of pipes.
- static FILE * `exe_popen` (const char *fname, const char *mode)
Helper function for opening a pipe from or to a given program.
- int `fileclose` (FILE *f)
Close a file or fifo but not if it is one of the standard streams.

- FILE * **fileopen** (const char *fname, const char *mode)
Search for a file in the include path list and open it if possible.
- static void **fileopen_env_init** (void)
- static FILE * **fopenx** (const char *fname, const char *mode)
- static void **freeexepath** ()
Free a whole list of execution path elements.
- static void **freepath** ()
Free a whole list of include path elements.
- struct **incpath** * **get_include_path** (void)
- void **initexepath** (const char *default_exe_path)
- void **initpath** (const char *default_path)
Init the path list, with default_path as the only entry.
- void **listpath** (char *buffer, size_t bufsize)
Show the list of include paths.
- static FILE * **popenx** (const char *fname, const char *mode)
- void **set_permissive_pipes** (int p)
Enable or disable the permissive execution of pipes.
- static FILE * **ssh_popen** (const char *fname, const char *mode, int compression)
Helper function for opening a file on a remote SSH server.
- static FILE * **uri_popen** (const char *fname, const char *mode, int compression)
Helper function for opening a file with a URI (<http://> etc.).

Variables

- static int **foei_done** = 0
- static int **parallel** = 0
- static int **permissive_pipes** = 0
Allow any execution pipe command if this variable is non-zero.
- static struct **incpath** * **root_exe_path** = NULL
The starting element for execution paths.
- static struct **incpath** * **root_path** = NULL
The starting element of include paths.
- static int **verbose** = 0
Use to decide if open/close success/failure is reported.
- static int **with_exec** = 1
- static int **with_fallback** = 1

6.6.1 Detailed Description

The functions provided in this file provide an enhanced replacement **fileopen()** for the C standard library's **fopen()** function. The enhancements are in several areas:

- Where possible files are opened such that more than 2 gigabytes of data can be accessed on 32-bit systems when suitably compiled. This also works with software where a '**-D_FILE_OFFSET_BITS=64**' at compile-time cannot be used (of which ROOT is an infamous example).
- For reading files, a list of paths can be configured before the the first **fileopen()** call and all files without absolute paths will be searched in these paths. Writing always strictly follows the given file name and will not search in the path list.

- Files compressed with `gzip` or `bzip2` can be handled on the fly. Files with corresponding file name extensions (`.gz` and `.bz2`) will be automatically decompressed when reading or compressed when writing (in a pipe, i.e. without producing temporary copies).
- In the same way, files compressed with `lzop` (for extension `.lzo`), `lzma` (for extension `.lzma`) as well as `xz` (for extension `@.xz`) and `lz4` (for extension `.lz4`) are handled on the fly. No check is made if these programs are installed.
- URIs (uniform resource identifiers) starting with `http:`, `https:`, or `ftp:` will also be opened in a pipe, with optional decompression, depending on the ending of the URI name. You can therefore easily process files located on a web or ftp server. Access is limited to reading.
- Files on any SSH server where you can login without a password can be read as `'ssh://user@host:filepath'` where filepath can be an absolute path (starting with `'/'`) or one relative to the users home directory.
- Input and output can also be from/to a user-defined program. Restrictions apply there which prevent execution of any program by default. Either a list of accepted execution paths has to be set up beforehand with `initexepath()/addexepath()` or permissive mode can be enabled, allowing execution of any given program.

Author

Konrad Bernloehr

Date

Nov. 2000

CVS \$Date: 2019/01/10 14:40:51 \$

Version

CVS \$Revision: 1.31 \$

6.6.2 Function Documentation**6.6.2.1 addexepath()**

```
void addexepath (
    const char * name )
```

The path name is always copied to a newly allocated memory location. This path name can actually be a colon-separated list, as for `initexepath()`.

References `addpath()`, `root_exe_path`, and `root_path`.

6.6.2.2 addpath()

```
void addpath (
    const char * name )
```

The path name is always copied to a newly allocated memory location. This path name can actually be a colon-separated list, as for [initpath\(\)](#). Also environment variables (indicated by starting with '\$', e.g. "\$HOME") are accepted (and may expand into colon-separated list) but no mixed expansion (like "\$HOME/bin").

References [getword\(\)](#), [incpath::next](#), [incpath::path](#), and [root_path](#).

Here is the caller graph for this function:



6.6.2.3 cmp_popen()

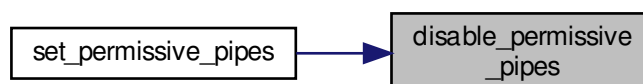
```
static FILE* cmp_popen (
    const char * fname,
    const char * mode,
    int compression ) [static]
```

6.6.2.4 disable_permissive_pipes()

```
void disable_permissive_pipes (
    void )
```

References [permissive_pipes](#).

Here is the caller graph for this function:



6.6.2.5 enable_permissive_pipes()

```
void enable_permissive_pipes (
    void )
```

References `permissive_pipes`, and `root_exe_path`.

Here is the caller graph for this function:



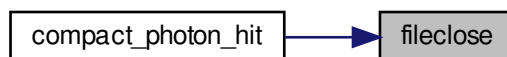
6.6.2.6 exe_popen()

```
static FILE* exe_popen (
    const char * fname,
    const char * mode ) [static]
```

6.6.2.7 fclose()

```
int fclose (
    FILE * f )
```

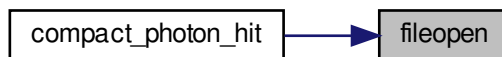
Here is the caller graph for this function:



6.6.2.8 `fileopen()`

```
FILE* fileopen (
    const char * fname,
    const char * mode )
```

Here is the caller graph for this function:



6.6.2.9 `freeexepath()`

```
static void freeexepath ( ) [static]
```

References `incpath::next`, `incpath::path`, and `root_exe_path`.

6.6.2.10 `freepath()`

```
static void freepath ( ) [static]
```

References `incpath::next`, `incpath::path`, and `root_path`.

6.6.2.11 `initpath()`

```
void initpath (
    const char * default_path )
```

6.6.2.12 `listpath()`

```
void listpath (
    char * buffer,
    size_t bufsize )
```

References `incpath::path`, and `root_path`.

6.6.2.13 set_permissive_pipes()

```
void set_permissive_pipes (
    int p )
```

References `disable_permissive_pipes()`, and `enable_permissive_pipes()`.

6.6.2.14 uri_popen()

```
static FILE * uri_popen (
    const char * fname,
    const char * mode,
    int compression ) [static]
```

6.6.3 Variable Documentation

6.6.3.1 permissive_pipes

```
int permissive_pipes = 0 [static]
```

6.6.3.2 root_exe_path

```
struct incpath* root_exe_path = NULL [static]
```

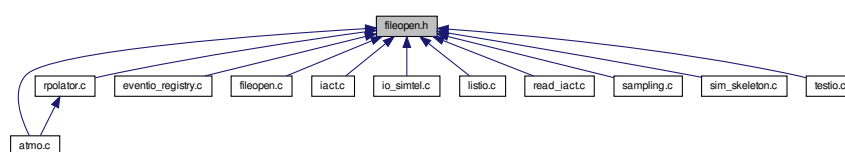
6.6.3.3 root_path

```
struct incpath* root_path = NULL [static]
```

6.7 fileopen.h File Reference

Function prototypes for `fileopen.c`.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [incpath](#)

An element in a linked list of include paths.

Functions

- void [addexepath](#) (const char *name)
Add a path to the list of execution paths, if not already there.
- void [addpath](#) (const char *name)
Add a path to the list of include paths, if not already there.
- void [disable_permissive_pipes](#) (void)
Disable the permissive execution of pipes.
- void [enable_permissive_pipes](#) (void)
Enable the permissive execution of pipes.
- int [fileclose](#) (FILE *f)
Close a file or fifo but not if it is one of the standard streams.
- FILE * [fileopen](#) (const char *fname, const char *mode)
Search for a file in the include path list and open it if possible.
- struct [incpath](#) * [get_include_path](#) (void)
- void [initexepath](#) (const char *default_path)
- void [initpath](#) (const char *default_path)
Init the path list, with default_path as the only entry.
- void [listpath](#) (char *buffer, size_t bufsize)
Show the list of include paths.
- void [set_permissive_pipes](#) (int p)
Enable or disable the permissive execution of pipes.

6.7.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2019/02/04 13:29:01 \$

Version

CVS \$Revision: 1.11 \$

6.7.2 Function Documentation

6.7.2.1 addexepath()

```
void addexepath (
    const char * name )
```

The path name is always copied to a newly allocated memory location. This path name can actually be a colon-separated list, as for `initexepath()`.

References `addpath()`, `root_exe_path`, and `root_path`.

6.7.2.2 addpath()

```
void addpath (
    const char * name )
```

The path name is always copied to a newly allocated memory location. This path name can actually be a colon-separated list, as for `initpath()`. Also environment variables (indicated by starting with '\$', e.g. "\$HOME") are accepted (and may expand into colon-separated list) but no mixed expansion (like "\$HOME/bin").

References `getword()`, `incpath::next`, `incpath::path`, and `root_path`.

Here is the caller graph for this function:

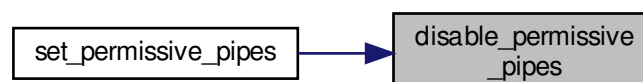


6.7.2.3 disable_permissive_pipes()

```
void disable_permissive_pipes (
    void )
```

References `permissive_pipes`.

Here is the caller graph for this function:



6.7.2.4 enable_permissive_pipes()

```
void enable_permissive_pipes (  
    void )
```

References `permissive_pipes`, and `root_exe_path`.

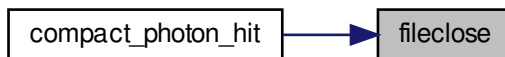
Here is the caller graph for this function:



6.7.2.5 fclose()

```
int fclose (  
    FILE * f )
```

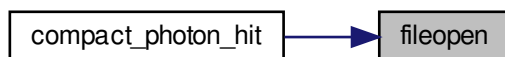
Here is the caller graph for this function:



6.7.2.6 fopen()

```
FILE* fopen (  
    const char * fname,  
    const char * mode )
```

Here is the caller graph for this function:



6.7.2.7 initpath()

```
void initpath (
    const char * default_path )
```

6.7.2.8 listpath()

```
void listpath (
    char * buffer,
    size_t bufsize )
```

References `incpath::path`, and `root_path`.

6.7.2.9 set_permissive_pipes()

```
void set_permissive_pipes (
    int p )
```

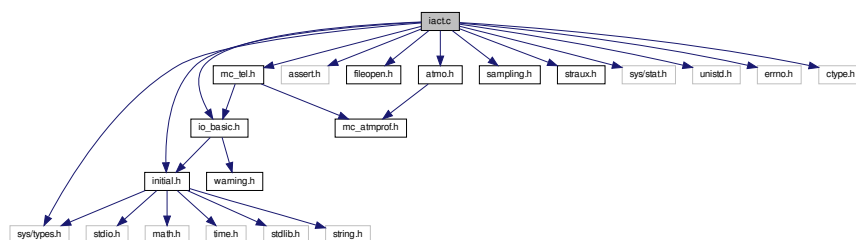
References `disable_permissive_pipes()`, and `enable_permissive_pipes()`.

6.8 iact.c File Reference

CORSIKA interface for Imaging Atmospheric Cherenkov Telescopes etc.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include <assert.h>
#include "fileopen.h"
#include "atmo.h"
#include "sampling.h"
#include "straux.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <ctype.h>
```

Include dependency graph for `iact.c`:



Data Structures

- struct [detstruct](#)
A structure describing a detector and linking its photons bunches to it.
- struct [gridstruct](#)

Macros

- #define [EXTERNAL_STORAGE](#) 1
Enable external temporary bunch storage.
- #define **EXTRA_MEM** 0
- #define **EXTRA_MEM_1** EXTRA_MEM
- #define **EXTRA_MEM_10** EXTRA_MEM
- #define **EXTRA_MEM_11** EXTRA_MEM
- #define **EXTRA_MEM_12** EXTRA_MEM
- #define **EXTRA_MEM_2** EXTRA_MEM
- #define **EXTRA_MEM_3** EXTRA_MEM
- #define **EXTRA_MEM_4** EXTRA_MEM
- #define **EXTRA_MEM_5** EXTRA_MEM
- #define **EXTRA_MEM_6** EXTRA_MEM
- #define **EXTRA_MEM_7** EXTRA_MEM
- #define **EXTRA_MEM_8** EXTRA_MEM
- #define **EXTRA_MEM_9** EXTRA_MEM
- #define [GRID_SIZE](#) 1000
unit: cm
- #define **HAVE_EVENTIO_FUNCTIONS** 1
- #define **IACT_ATMO_VERSION** "1.60 (2019-07-24)"
- #define [INTERNAL_LIMIT](#) 100000
Start external storage after so many bunches.
- #define **M_PI** 3.14159265358979323846 /* pi */
- #define **max**(a, b) ((a)>(b)?(a):(b))
- #define [MAX_ARRAY_SIZE](#) 1000 /* Use the same limit as in CORSIKA itself. */
Maximum number of telescopes (or other detectors) per array.
- #define **MAX_BUNCHES** 5000000
- #define **MAX_CLASS** 1
- #define **MAX_IO_BUFFER** 200000000
- #define **min**(a, b) ((a)<(b)?(a):(b))
- #define [NBUNCH](#) 5000
Memory allocation step size for bunches.
- #define **PIPE_OUTPUT** 1
- #define **PRMPAR_SIZE** 17
- #define **square**(x) ((x)*(x))
- #define **UNKNOWN_LONG_DIST** 1

Typedefs

- typedef float [cors_real_t](#)
*Type for CORSIKA floating point numbers remaining REAL*4.*

Functions

- static int `compact_photon_hit` (struct `detstruct` *det, double x, double y, double cx, double cy, double sx, double sy, double photons, double ctime, double zem, double lambda)
Store a photon bunch for a given telescope in compact format.
- static void `cross_prod` (double *v1, double *v2, double *v3)
Cross (outer) product of two 3-D vectors v1, v2 into 3-D vector v3.
- static int `expand_env` (char *fname, size_t maxlen)
Expanding environment variables ourselves rather than passing that on a shell later, so that we can still check characters after expansion.
- void `extprim_setup` (const char *text)
Placeholder function for activating and setting up user-defined (external to CORSIKA) controlled over types, spectra, and angular distribution of primaries.
- void `extprm_` (cors_dbl_t *type, cors_dbl_t *eprim, double *thetap, double *phip)
Placeholder function for external shower-by-shower setting of primary type, energy, and direction.
- void `get_iact_stats` (long *sb, double *ab, double *ap)
- void `get_impact_offset` (cors_real_t evth[273], cors_dbl_t prmpar[PRMPAR_SIZE])
Approximate impact offset of primary due to geomagnetic field.
- static void `get_mass_charge` (int type, double *mass, double *charge)
- double `heigh_` (double *thickness)
The CORSIKA built-in function for the height as a function of overburden.
- static void `iact_param` (const char *text0)
Processing of IACT module specific parameters in Corsika input.
- double `iact_rndm` (int dummy)
- static int `in_detector` (struct `detstruct` *det, double x, double y, double sx, double sy)
Check if a photon bunch hits a particular telescope volume.
- static void `ioerrorcheck` ()
- static int `is_off` (char *word)
- static int `is_on` (char *word)
- static int `Nint_f` (double x)
Nearest integer function.
- static double `norm3` (double *v)
Norm of a 3-D vector.
- static void `norm_vec` (double *v)
Normalize a 3-D vector.
- static int `photon_hit` (struct `detstruct` *det, double x, double y, double cx, double cy, double sx, double sy, double photons, double ctime, double zem, double lambda)
Store a photon bunch for a given telescope in long format.
- double `refidx_` (double *height)
Index of refraction as a function of altitude [cm].
- double `rhof_` (double *height)
The CORSIKA built-in density lookup function.
- void `rmmard_` (double *, int *, int *)
- static double `rndm` (int dummy)
Random number interface using sequence 4 of CORSIKA.
- void `sample_offset` (const char *sampling_fname, double core_range, double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, double *xoff, double *yoff, double *sampling_area)
Get uniformly sampled or importance sampled offset of array with respect to core, in the plane perpendicular to the shower axis.
- static int `set_random_systems` (double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, int volflag)
Randomly scatter each array of detectors in given area.

- void `set_system_displacement` (double dx, double dy)
Report displaced core positions in event header.
- void `telasu_` (int *n, `cors_dbl_t` *dx, `cors_dbl_t` *dy)
Setup how many times each shower is used.
- void `telend_` (`cors_real_t` evte[273])
End of event.
- void `televt_` (`cors_real_t` evth[273], `cors_dbl_t` prmpar[PRMPAR_SIZE])
Start of new event.
- void `telfil_` (const char *name0)
Define the output file for photon bunches hitting the telescopes.
- void `telinf_` (int *itel, double *x, double *y, double *z, double *r, int *exists)
Return information about configured telescopes back to CORSIKA.
- void `telling_` (int *type, double *data, int *ndim, int *np, int *nthick, double *thickstep)
Write CORSIKA 'longitudinal' (vertical) distributions.
- void `tellni_` (const char *line, int *llength)
Keep a record of CORSIKA input lines.
- int `telout_` (`cors_dbl_t` *bsize, `cors_dbl_t` *wt, `cors_dbl_t` *px, `cors_dbl_t` *py, `cors_dbl_t` *pu, `cors_dbl_t` *pv, `cors_dbl_t` *ctime, `cors_dbl_t` *zem, `cors_dbl_t` *lambda)
Check if a photon bunch hits one or more simulated detector volumes.
- void `telrne_` (`cors_real_t` rune[273])
Write run end block to the output file.
- void `telrnh_` (`cors_real_t` runh[273])
Save aparameters from CORSIKA run header.
- void `telset_` (`cors_dbl_t` *x, `cors_dbl_t` *y, `cors_dbl_t` *z, `cors_dbl_t` *r)
Add another telescope to the system (array) of telescopes.
- void `telshw_` ()
Show what telescopes have actually been set up.
- void `telsmp_` (const char *name)

Variables

- static double `airlightspeed` = 29.9792458/1.0002256
[cm/ns] at H=2200 m
- static double `all_bunches`
- static double `all_bunches_run`
- static double `all_photons`
- static double `all_photons_run`
- static double `arr_cosang`
- static double `arr_sinang`
- static double `arrang`
- static double `atmo_top`
- static double `Bfield` [3]
Magnetic field vector in detector coordinate system (with Bz positive if upwards)
- static double `bxplane` [3]
- static double `byplane` [3]
Spanning vectors of shower plane such that projection of B field is in bxplane direction.
- static double `core_range`
The maximum core offset of array centres in circular distribution.
- static double `core_range1`
The maximum core offsets in x,y for rectangular distribution.
- static double `core_range2`

- int **cors_4dig_def** = 6900
- double **cors_ver_def** = 6.900
- static struct [linked_string](#) **corsika_inputs** = { corsika_inputs_head, NULL }
- static char **corsika_inputs_head** [80]
- int **corsika_version** = ([CORSIKA_VERSION](#))

The CORSIKA version actually running.

- static int **count_print_evt** = 0
- static int **count_print_tel** = 0
- static int **det_in_class** [MAX_CLASS]
- static struct [detstruct](#) ** **detector**
- static double **dmax** = 0.

Max.

- static int **do_print**
- static double **energy**
- static int **event_number**
- static double **first_int**
- static struct [gridstruct](#) * **grid**
- static int **grid_elements**
- static int **grid_nx**
- static int **grid_ny**
- static double **grid_x_high**
- static double **grid_x_low**
- static double **grid_y_high**
- static double **grid_y_low**
- static int **impact_correction** = 1

Correct impact position if non-zero.

- static double **impact_offset** [2]

Offset of impact position of charged primaries.

- static [IO_BUFFER](#) * **iobuf**
- static double **lambda1**
- static double **lambda2**
- static long **max_bunches** = [MAX_BUNCHES](#)
- static int **max_internal_bunches** = [INTERNAL_LIMIT](#)

The largest number of photon bunches kept in main memory before attempting to flush them to temporary files on disk.

- static size_t **max_io_buffer** = [MAX_IO_BUFFER](#)

The largest block size in the output data, which must hold all photons bunches of one array.

- static int **max_print_evt** = 100
- static int **max_print_tel** = 10
- static int **narray**
- static int * **ndet**
- static int **nevents**
- static int **nrun**
- static int **nsys** = 1

Number of arrays.

- static int **ntel** = 0

Number of telescopes set up.

- static double **obs_dx**
- static double **obs_dy**
- static double **obs_height**
- static double **off_axis**
- static char * **output_fname** = NULL

The name of the output file for eventio format data.

- static char * **output_options** = NULL
- static double **phi_central**
- static double **phi_prim**
- static double **pprim** [3]
Momentum vector of primary particle.
- static int **primary**
- static double **raise_tel**
Non-zero if any telescope has negative z.
- static double **rmax** = 0.
Max.
- static double **rtel** [MAX_ARRAY_SIZE]
- static char * **sampling_fname**
The name of the file providing parameters for importance sampling.
- static int **skip_off2** = 1
- static int **skip_print** = 1
- static int **skip_print2** = 100
- static long **stored_bunches**
- static int **tel_individual** = 0
- static size_t **tel_split_threshold** = 10000000
- static int **televt_done**
- static double **theta_central**
The central value of the allowed ranges in theta and phi.
- static double **theta_prim**
- static double **toffset**
- static int **use_compact_format** = 1
- static double **ush**
- static double **ushc**
- static double **vsh**
- static double **vshc**
- static double * **weight**
- int **with_extprim** = 0
- static double **wsh**
- static double **wshc**
- static double **xdisplaced** = 0.
Extra reported core offset displacement (not used with CORSIKA)
- static double * **xoffset**
- static double **xtel** [MAX_ARRAY_SIZE]
Position and size definition of fiducial spheres.
- static double **ydisplaced** = 0.
- static double * **yoffset**
- static double **ytel** [MAX_ARRAY_SIZE]
- static double **ztel** [MAX_ARRAY_SIZE]

6.8.1 Detailed Description

Author

Konrad Bernloehr

Date

```
CVS $Date: 2019/07/24 12:25:09 $
```

```
CVS $Revision: 1.122 $
```

Version 1.2.44 (for IACT/ATMO package version 1.60)

This file implements a CORSIKA interface for the simulation of (3-D) arrays of Cherenkov telescopes. A whole array may be simulated in multiple instances with random offsets of each instance. For full use of this software additional files are required which are available now on request from Konrad Bernloehr (e-mail: Konrad.Bernloehr@mpi-hd.mpg.de). These additional files should be included in the same add-on package to CORSIKA which includes this file. A fallback mechanism is included to use the normal CORSIKA output of Cherenkov photon bunches instead of the dedicated output functions from the unavailable files. However, this fallback mechanism has important drawbacks: information about positions of telescopes are completely lost and no photon bunches are collected in memory because the collected bunches would never be written out. For those reasons you are advised to obtain and use the additional software.

General comments on this file:

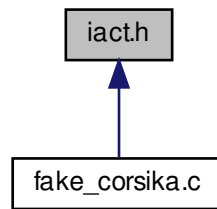
Routines provided in this file interface to recent versions of the CORSIKA air shower simulation program (the FORTRAN implementation only for now, not CORSIKA 8). Modifications to CORSIKA have been kept as simple as possible and the existing routines for production of Cherenkov light have been largely maintained. Setup of the telescope systems to be simulated is via the usual CORSIKA input file (the syntax of which has been extended by a few additional keywords). These telescope systems can be randomly scattered several times within a given area. All treatment whether a bunch of photons hits a telescope is done by the routines in this file. Photon bunches are kept in main memory until the end of the event. This might be a limitation when simulating large showers / many telescopes / many systems of telescopes on a computer with little memory. An option to store photon bunches in a temporary file has, therefore, been included. After the end of an event in CORSIKA all photon bunches (sorted by system and telescope) are written to a data file in the 'eventio' portable data format also used for CRT and HEGRA CT data. All CORSIKA run/event header/trailer blocks are also written to this file.

This version comes with sections for conditional compilation like EXTENDED_TELOUT CORSIKA is compiled with extended interface (IACTEXT option). Information about particles on ground is stored. IACTEXT For all practical purposes a synonym to EXTENDED_TELOUT. STORE_EMITTER Store information about all particles emitting light after the photon bunch. This duplicated the amount of data. Requires the IACTEXT option to be activated. MARK_DIRECT_LIGHT The Cherenkov photons bunches from the primary particle and its leading/major fragments are marked up with a non-zero wavelength (1: direct, 2, 3: major fragments). Requires the IACTEXT option to be activated. See the ALL_WL_RANDOM configuration parameter to sim_telarray. CORSIKA_SAVES_PHOTONS CORSIKA should save photons in its own format. IACT_NO_GRID For optimization of finding detector hits in normal simulations a detector is assigned to each grid cell onto which the detector throws a 'shadow', reducing the detector intersection checks a lot. In special shower simulations (example: a 10000 m high string of telescopes) or in technical simulations (e.g. using the IACT interface for artificial light sources, without CORSIKA) this may fail and explicit intersection of each photon bunch with each detector sphere needs to be checked. NO_EXTERNAL_ATMOSPHERES CORSIKA was compiled without the ATMEXT option and we do not handle tables of atmospheric density etc. Note that these extensions may not have been tested in a long time. Use with care.

6.9 iact.h File Reference

Function declarations for CORSIKA IACT interface.

This graph shows which files directly or indirectly include this file:



- `#define PRMPAR_SIZE 17`
- `/`
- `typedef float cors_real_t`
*Type for CORSIKA floating point numbers remaining REAL*4.*
- `typedef double cors_dbl_t`
*Type for CORSIKA numbers which are currently REAL*8.*
- `void get_iact_stats (long *sb, double *ab, double *ap)`
- `double iact_rndm (int dummy)`
- `void ioerrorcheck (void)`
- `void telfil_ (const char *name)`
Define the output file for photon bunches hitting the telescopes.
- `void telsmp_ (const char *name)`
- `void telshw_ (void)`
Show what telescopes have actually been set up.
- `void telinf_ (int *itel, double *x, double *y, double *z, double *r, int *exists)`
Return information about configured telescopes back to CORSIKA.
- `void telrnh_ (cors_real_t runh[273])`
Save aparameters from CORSIKA run header.
- `void tellni_ (const char *line, int *llength)`
Keep a record of CORSIKA input lines.
- `void telrne_ (cors_real_t rune[273])`
Write run end block to the output file.
- `void telasu_ (int *n, cors_dbl_t *dx, cors_dbl_t *dy)`
Setup how many times each shower is used.
- `void telset_ (cors_dbl_t *x, cors_dbl_t *y, cors_dbl_t *z, cors_dbl_t *r)`
Add another telescope to the system (array) of telescopes.
- `void get_impact_offset (cors_real_t evth[273], cors_dbl_t prmpar[17])`
- `void televt_ (cors_real_t evth[273], cors_dbl_t prmpar[17])`
- `int telout_ (cors_dbl_t *bsize, cors_dbl_t *wt, cors_dbl_t *px, cors_dbl_t *py, cors_dbl_t *pu, cors_dbl_t *pv, cors_dbl_t *ctime, cors_dbl_t *zem, cors_dbl_t *lambda)`
Check if a photon bunch hits one or more simulated detector volumes.
- `void telling_ (int *type, double *data, int *ndim, int *np, int *nthick, double *thickstep)`
Write CORSIKA 'longitudinal' (vertical) distributions.
- `void telend_ (cors_real_t evte[273])`
End of event.
- `void extprim_setup (const char *text)`

Placeholder function for activating and setting up user-defined (external to CORSIKA) controlled over types, spectra, and angular distribution of primaries.

- void `extprm_` (`cors_dbl_t` *type, `cors_dbl_t` *eprim, double *thetap, double *pkip)

Placeholder function for external shower-by-shower setting of primary type, energy, and direction.

- void `set_system_displacement` (double dx, double dy)

Report displaced core positions in event header.

- double `refidx_` (double *height)

Index of refraction as a function of altitude [cm].

- double `rhof_` (double *height)

The CORSIKA built-in density lookup function.

- int `iact_check_track_` (double *x1, double *y1, double *z1, double *t1, double *e1, double *eta1, double *x2, double *y2, double *z2, double *t2, double *e2, double *eta2, double *amass, double *charge, double *wtthin)

6.9.1 Detailed Description

Function declarations are only needed if linking a C or C++ program to the IACT interface. The CORSIKA Fortran code is not using it.

Author

Konrad Bernloehr

Date

CVS \$Date: 2019/02/15 14:42:49 \$

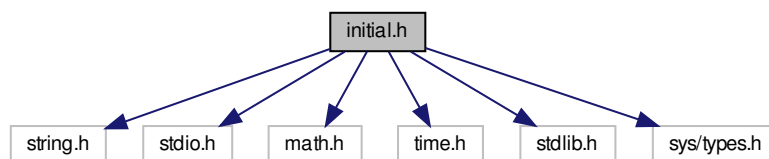
CVS \$Revision: 1.12 \$

6.10 initial.h File Reference

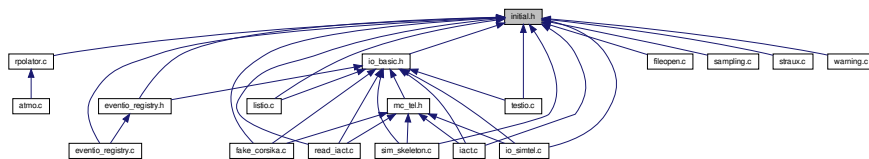
Identification of the system and including some basic include file.

```
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <sys/types.h>
```

Include dependency graph for initial.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define Abs(a) (((a)>=0)?(a):(-1*(a)))`
- `#define APPEND_BINARY "a"`
- `#define APPEND_TEXT "a"`
- `#define ARGLIST(a) a`
- `#define CONST_QUAL`
- `#define IEEE_FLOAT_FORMAT 1`
- `#define M_PI 3.14159265358979323846`
- `#define Max(a, b) ((a)>(b)?(a):(b))`
- `#define max(a, b) ((a)>(b)?(a):(b))`
- `#define Min(a, b) ((a)<(b)?(a):(b))`
- `#define min(a, b) ((a)<(b)?(a):(b))`
- `#define Nint(a) (((a)>=0.?)?((long)(a+0.5)):((long)(a-0.5)))`
- `#define READ_BINARY "r"`
- `#define READ_TEXT "r"`
- `#define REGISTER register`
- `#define SEEK_CUR 1`
- `#define WRITE_BINARY "w"`
- `#define WRITE_TEXT "w"`

Typedefs

- `typedef short int16_t`
- `typedef int int32_t`
- `typedef char int8_t`
- `typedef long intmax_t`
- `typedef unsigned short uint16_t`
- `typedef unsigned int uint32_t`
- `typedef unsigned char uint8_t`
- `typedef unsigned long uintmax_t`

6.10.1 Detailed Description

Author

Konrad Bernloehr

Date

1991 to 2010

\$Date: 2018/11/26 12:29:34 \$

Version

```
$Revision: 1.20 $
```

This file identifies a range of supported operating systems and processor types. As a result, some preprocessor definitions are made. A basic set of system include files (which may vary from one system to another) are included. In addition, compatibility between different systems is improved, for example between K&R compiler systems and ANSI C compilers of various flavours.

Identification of the host operating system (not CPU):

Supported identifiers are

OS_MSDOS

OS_VAXVMS

OS_UNIX

+ variant identifiers like

OS_ULTRIX, OS_LYNX, OS_LINUX, OS_DECUNIX, OS_AIX, OS_HPUX,
OS_DARWIN (Mac OS X).

Note: ULTRIX may be on VAX or MIPS, LINUX on Intel or Alpha,
OS_LYNX on 68K or PowerPC.

OS_OS9

You might first reset all identifiers here.

Then set one or more identifiers according to the system.

Identification of the CPU architecture:

Supported CPU identifiers are

CPU_I86

CPU_X86_64

CPU_VAX

CPU_MIPS

CPU_ALPHA

CPU_68K

CPU_RS6000

CPU_PowerPC

CPU_HPPA

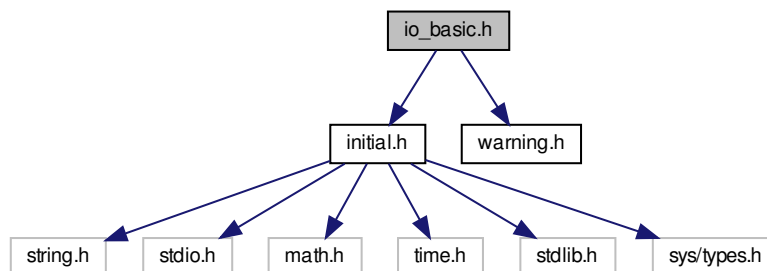
6.11 io_basic.h File Reference

Basic header file for eventio data format.

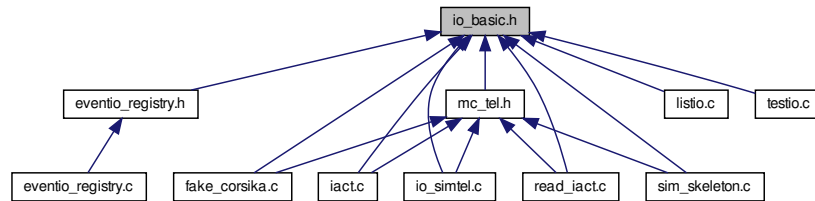
```
#include "initial.h"
```

```
#include "warning.h"
```

Include dependency graph for io_basic.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [_struct_IO_BUFFER](#)

The IO_BUFFER structure contains all data needed the manage the stuff.

- struct [_struct_IO_ITEM_HEADER](#)

An IO_ITEM_HEADER is to access header info for an I/O block and as a handle to the I/O buffer.

- struct [ev_reg_entry](#)

Macros

- #define **COPY_BYTES**(_target, _source, _num) memcpy(_target, _source, _num)
- #define **COPY_BYTES_SWAB**(_target, _source, _num) swab(_source, _target, _num)
- #define **EVENTIO_EXTENSION_FLAG** 2
- #define **EVENTIO_USER_FLAG** 1
- #define **get_byte**(p) (--(p)->r_remaining>=0? *(p)->data++ : -1)
- #define **get_bytes**(p, k) (((p)->r_remaining -= k) >=0 ? (*(p)->data+=k)-k : -1)
- #define **get_vector_of_int16** [get_vector_of_short](#)
- #define **get_vector_of_uint8** [get_vector_of_byte](#)
- #define **HAVE_EVENTIO_EXTENDED_LENGTH** 1
- #define **HAVE_EVENTIO_HEADER_LENGTH** 1
- #define **HAVE_EVENTIO_USER_FLAG** 1
- #define **IO_BUFFER_INITIAL_LENGTH** 32768L
- #define **IO_BUFFER_LENGTH_INCREMENT** 65536L
- #define **IO_BUFFER_MAXIMUM_LENGTH** 3000000L
- #define **MAX_IO_ITEM_LEVEL** 20
- #define **put_byte**(_c, _p)
- #define **put_vector_of_int16** [put_vector_of_short](#)
- #define **put_vector_of_uint8** [put_vector_of_byte](#)

Typedefs

- typedef unsigned char **BYTE**
- typedef struct [ev_reg_entry](#) [*\(EVREGSEARCH\)](#) (unsigned long t)
- typedef struct [_struct_IO_BUFFER](#) **IO_BUFFER**
- typedef struct [_struct_IO_ITEM_HEADER](#) **IO_ITEM_HEADER**
- typedef int [*\(IO_USER_FUNCTION\)](#) (unsigned char *, long, int)

Functions

- `IO_BUFFER * allocate_io_buffer (size_t buflen)`
Dynamic allocation of an I/O buffer.
- `int append_io_block_as_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, BYTE *_buffer, long length)`
Append data from one I/O block into another one.
- `int copy_item_to_io_block (IO_BUFFER *iobuf2, IO_BUFFER *iobuf, const IO_ITEM_HEADER *item_header)`
Copy a sub-item to another I/O buffer as top-level item.
- `double dbl_from_sfloat (const uint16_t *snum)`
Convert from the internal representation of an OpenGL 16-bit floating point number back to normal floating point representation.
- `void dbl_to_sfloat (double dnum, uint16_t *snum)`
Convert a double to the internal representation of a 16 bit floating point number as specified in the OpenGL 3.1 standard, also called a half-float.
- `const char * eventio_registered_description (unsigned long type)`
Extract the optional description for a given type number, if available.
- `const char * eventio_registered_typename (unsigned long type)`
This functions using the stored function pointer are now in the core eventio code.
- `int extend_io_buffer (IO_BUFFER *iobuf, unsigned next_byte, long increment)`
Extend the dynamically allocated I/O buffer.
- `struct ev_reg_entry * find_ev_reg (unsigned long t)`
This optionally available function is implemented externally.
- `int find_io_block (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)`
Find the beginning of the next I/O data block in the input.
- `void fltp_to_sfloat (const float *fnum, uint16_t *snum)`
Convert a float to the internal representation of a 16 bit floating point number as specified in the OpenGL 3.1 standard, also called a half-float.
- `void free_io_buffer (IO_BUFFER *iobuf)`
Free an I/O buffer that has been allocated at run-time.
- `uintmax_t get_count (IO_BUFFER *iobuf)`
Get an unsigned integer of unspecified length from an I/O buffer.
- `uint16_t get_count16 (IO_BUFFER *iobuf)`
Get an unsigned 16 bit integer of unspecified length from an I/O buffer.
- `uint32_t get_count32 (IO_BUFFER *iobuf)`
Get an unsigned 32 bit integer of unspecified length from an I/O buffer.
- `double get_double (IO_BUFFER *iobuf)`
Get a double from the I/O buffer.
- `int32_t get_int32 (IO_BUFFER *iobuf)`
Read a four byte integer from an I/O buffer.
- `int get_item_begin (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)`
Begin reading an item.
- `int get_item_end (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)`
End reading an item.
- `long get_long (IO_BUFFER *iobuf)`
Get 4-byte integer from I/O buffer and return as a long int.
- `int get_long_string (char *s, int nmax, IO_BUFFER *iobuf)`
Get a long string of ASCII characters from an I/O buffer.
- `double get_real (IO_BUFFER *iobuf)`
Get a floating point number (as written by put_real) from the I/O buffer.
- `intmax_t get_scount (IO_BUFFER *iobuf)`

- Get a signed integer of unspecified length from an I/O buffer.*

 - `int16_t get_scount16 (IO_BUFFER *iobuf)`

Shortened version of `get_scount` for up to 16 bits of data.
 - `int32_t get_scount32 (IO_BUFFER *iobuf)`

Shortened version of `get_scount` for up to 32 bits of data.
 - `double get_sfloat (IO_BUFFER *iobuf)`

Get a 16-bit float from an I/O buffer and expand it to a double.
 - `int get_short (IO_BUFFER *iobuf)`

Get a two-byte integer from an I/O buffer.
 - `int get_string (char *s, int nmax, IO_BUFFER *iobuf)`

Get a string of ASCII characters from an I/O buffer.
 - `uint16_t get_uint16 (IO_BUFFER *iobuf)`

Get one unsigned short from an I/O buffer.
 - `uint32_t get_uint32 (IO_BUFFER *iobuf)`

Get a four-byte unsigned integer from an I/O buffer.
 - `int get_var_string (char *s, int nmax, IO_BUFFER *iobuf)`

Get a string of ASCII characters from an I/O buffer.
 - `void get_vector_of_byte (BYTE *vec, int num, IO_BUFFER *iobuf)`

Get a vector of bytes from an I/O buffer.
 - `void get_vector_of_double (double *vec, int num, IO_BUFFER *iobuf)`

Get a vector of floating point numbers as 'doubles' from an I/O buffer.
 - `void get_vector_of_float (float *vec, int num, IO_BUFFER *iobuf)`

Get a vector of floating point numbers as 'floats' from an I/O buffer.
 - `void get_vector_of_int (int *vec, int num, IO_BUFFER *iobuf)`

Get a vector of (small) integers from I/O buffer.
 - `void get_vector_of_int32 (int32_t *vec, int num, IO_BUFFER *iobuf)`

Get a vector of 32 bit integers from I/O buffer.
 - `void get_vector_of_int_scount (int *vec, int num, IO_BUFFER *iobuf)`

Get an array of ints as `scount32` data from an I/O buffer.
 - `void get_vector_of_long (long *vec, int num, IO_BUFFER *iobuf)`

Get a vector of 4-byte integers as long int from I/O buffer.
 - `void get_vector_of_real (double *vec, int num, IO_BUFFER *iobuf)`

Get a vector of floating point numbers as 'doubles' from an I/O buffer.
 - `void get_vector_of_short (short *vec, int num, IO_BUFFER *iobuf)`

Get a vector of short integers from I/O buffer.
 - `void get_vector_of_uint16 (uint16_t *uval, int num, IO_BUFFER *iobuf)`

Get a vector of unsigned shorts from an I/O buffer.
 - `void get_vector_of_uint16_scount_differential (uint16_t *vec, int num, IO_BUFFER *iobuf)`

Get an array of `uint16_t` as differential `scount` data from an I/O buffer.
 - `void get_vector_of_uint32 (uint32_t *vec, int num, IO_BUFFER *iobuf)`

Get a vector of 32 bit integers from I/O buffer.
 - `void get_vector_of_uint32_scount_differential (uint32_t *vec, int num, IO_BUFFER *iobuf)`

Get an array of `uint32_t` as differential `scount` data from an I/O buffer.
 - `int list_io_blocks (IO_BUFFER *iobuf, int verbosity)`

Show the top-level item of an I/O block on standard output.
 - `int list_sub_items (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, int maxlevel, int verbosity)`

Display the contents of sub-items on standard output.
 - `long next_subitem_ident (IO_BUFFER *iobuf)`

Reads the header of a sub-item and return the identifier of it.
 - `long next_subitem_length (IO_BUFFER *iobuf)`

Reads the header of a sub-item and return the length of it.

- int `next_subitem_type` (`IO_BUFFER` *iobuf)
Reads the header of a sub-item and return the type of it.
- void `put_count` (uintmax_t num, `IO_BUFFER` *iobuf)
Put an unsigned integer of unspecified length to an I/O buffer.
- void `put_count16` (uint16_t num, `IO_BUFFER` *iobuf)
Shortened version of put_count for up to 16 bits of data.
- void `put_count32` (uint32_t num, `IO_BUFFER` *iobuf)
Shortened version of put_count for up to 32 bits of data.
- void `put_double` (double d, `IO_BUFFER` *iobuf)
Put a 'double' as such into an I/O buffer.
- void `put_int32` (int32_t num, `IO_BUFFER` *iobuf)
Write a four-byte integer to an I/O buffer.
- int `put_item_begin` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Begin putting another (sub-) item into the output buffer.
- int `put_item_begin_with_flags` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header, int user_flag, int extended)
Begin putting another (sub-) item into the output buffer.
- int `put_item_end` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
End of putting an item into the output buffer.
- void `put_long` (long num, `IO_BUFFER` *iobuf)
Put a four-byte integer taken from a 'long' into an I/O buffer.
- int `put_long_string` (const char *s, `IO_BUFFER` *iobuf)
Put a long string of ASCII characters into an I/O buffer.
- void `put_real` (double d, `IO_BUFFER` *iobuf)
Put a 4-byte floating point number into an I/O buffer.
- void `put_scount` (intmax_t num, `IO_BUFFER` *iobuf)
Put a signed integer of unspecified length to an I/O buffer.
- void `put_scount16` (int16_t num, `IO_BUFFER` *iobuf)
Shorter version of put_scount for up to 16 bytes of data.
- void `put_scount32` (int32_t num, `IO_BUFFER` *iobuf)
Shorter version of put_scount for up to 32 bytes of data.
- void `put_sfloat` (double dnum, `IO_BUFFER` *iobuf)
Put a 16-bit float to an I/O buffer.
- void `put_short` (int num, `IO_BUFFER` *iobuf)
Put a two-byte integer on an I/O buffer.
- int `put_string` (const char *s, `IO_BUFFER` *iobuf)
Put a string of ASCII characters into an I/O buffer.
- void `put_uint32` (uint32_t num, `IO_BUFFER` *iobuf)
Put a four-byte integer into an I/O buffer.
- int `put_var_string` (const char *s, `IO_BUFFER` *iobuf)
Put a string of ASCII characters into an I/O buffer.
- void `put_vector_of_byte` (const BYTE *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of bytes into an I/O buffer.
- void `put_vector_of_double` (const double *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of doubles into an I/O buffer.
- void `put_vector_of_float` (const float *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of floats as IEEE 'float' numbers into an I/O buffer.
- void `put_vector_of_int` (const int *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of integers (range -32768 to 32767) into I/O buffer.
- void `put_vector_of_int32` (const int32_t *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of 32 bit integers into I/O buffer.

- void `put_vector_of_int_scount` (const int *vec, int num, `IO_BUFFER` *iobuf)
Put an array of ints as scount32 data into an I/O buffer.
- void `put_vector_of_long` (const long *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of long int as 4-byte integers into an I/O buffer.
- void `put_vector_of_real` (const double *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of doubles as IEEE 'float' numbers into an I/O buffer.
- void `put_vector_of_short` (const short *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of 2-byte integers on an I/O buffer.
- void `put_vector_of_uint16` (const uint16_t *uval, int num, `IO_BUFFER` *iobuf)
Put a vector of unsigned shorts into an I/O buffer.
- void `put_vector_of_uint16_scount_differential` (uint16_t *vec, int num, `IO_BUFFER` *iobuf)
Put an array of uint16_t as differential scount data into an I/O buffer.
- void `put_vector_of_uint32` (const uint32_t *vec, int num, `IO_BUFFER` *iobuf)
Put a vector of 32 bit integers into I/O buffer.
- void `put_vector_of_uint32_scount_differential` (uint32_t *vec, int num, `IO_BUFFER` *iobuf)
Put an array of uint16_t as differential scount data into an I/O buffer.
- int `read_io_block` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Read the data of an I/O block from the input.
- int `remove_item` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Remove an item from an I/O buffer.
- int `reset_io_block` (`IO_BUFFER` *iobuf)
Reset an I/O block to its empty status.
- int `rewind_item` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Go back to the beginning of an item.
- int `search_sub_item` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header, `IO_ITEM_HEADER` *sub_↵
item_header)
Search for an item of a specified type.
- void `set_eventio_registry_hook` (EVREGSEARCH fptr)
This function should be used to set the find_ev_reg_ptr function pointer.
- int `skip_io_block` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Skip the data of an I/O block from the input.
- int `skip_subitem` (`IO_BUFFER` *iobuf)
When the next sub-item is of no interest, it can be skipped.
- int `unget_item` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Go back to the beginning of an item being read.
- int `unput_item` (`IO_BUFFER` *iobuf, `IO_ITEM_HEADER` *item_header)
Undo writing at the present level.
- int `write_io_block` (`IO_BUFFER` *iobuf)
Write an I/O block to the block's output.

6.11.1 Detailed Description

Author

Konrad Bernloehr

Date

1991 to 2014

CVS \$Date: 2019/02/20 14:50:15 \$

Version

CVS \$Revision: 1.27 \$

Header file for structures and function prototypes for the basic eventio functions. Not to be used to declare any project-specific structures and prototypes! Declare any such things in 'io_project.h' or in separate header files.

6.11.2 Macro Definition Documentation

6.11.2.1 put_byte

```
#define put_byte(
    _c,
    _p )
```

Value:

```
(-(_p)->w_remaining>=0 ? \
    (*(_p)->data++ = (BYTE) (_c)) : \
    (BYTE) extend\_io\_buffer(_p, (unsigned) (_c), \
        (IO_BUFFER_LENGTH_INCREMENT)))
```

6.11.3 Function Documentation

6.11.3.1 allocate_io_buffer()

```
IO\_BUFFER* allocate_io_buffer (
    size_t buflen )
```

Dynamic allocation of an I/O buffer. The actual length of the buffer is passed as an argument. The buffer descriptor is initialized.

Parameters

<i>buflen</i>	The length of the actual buffer in bytes. A safety margin of 4 bytes is added.
---------------	--

Returns

Pointer to I/O buffer or NULL if allocation failed.

6.11.3.2 append_io_block_as_item()

```
int append_io_block_as_item (
    IO\_BUFFER * iobuf,
    IO\_ITEM\_HEADER * item_header,
    BYTE * buffer,
    long length )
```

Append the data from a complete i/o block as an additional subitem to another i/o block.

Parameters

<i>iobuf</i>	The target I/O buffer descriptor, must be 'opened' for 'writing', i.e. ' put_item_begin() ' must be called.
<i>item_header</i>	Item header of the item in iobuf which is currently being filled.
<i>buffer</i>	Data to be filled in. Must be all data from an I/O buffer, including the 4 signature bytes.
<i>length</i>	The length of buffer in bytes.

Returns

0 (o.k.), -1 (error), -2 (not enough memory etc.)

6.11.3.3 copy_item_to_io_block()

```
int copy_item_to_io_block (
    IO_BUFFER * iobuf2,
    IO_BUFFER * iobuf,
    const IO_ITEM_HEADER * item_header )
```

Parameters

<i>iobuf2</i>	Target I/O buffer descriptor.
<i>iobuf</i>	Source I/O buffer descriptor.
<i>item_header</i>	Header for the item in iobuf that should be copied to iobuf2.

Returns

0 (o.k.), -1 (error), -2 (not enough memory etc.)

6.11.3.4 dbl_to_sfloat()

```
void dbl_to_sfloat (
    double dnum,
    uint16_t * snum )
```

This is done via an intermediate float representation.

Parameters

<i>dnum</i>	The number to be converted.
<i>snum</i>	Pointer for the resulting representation, as stored in an unsigned 16-bit integer (1 bit sign, 5 bits exponent, 10 bits mantissa).

References fltp_to_sfloat().

6.11.3.5 eventio_registered_typename()

```
const char* eventio_registered_typename (
    unsigned long type )
```

This functions using the stored function pointer are now in the core eventio code.

References find_ev_reg(), ev_reg_entry::name, and ev_reg_entry::type.

6.11.3.6 extend_io_buffer()

```
int extend_io_buffer (
    IO_BUFFER * iobuf,
    unsigned next_byte,
    long increment )
```

Extend the dynamically allocated I/O buffer and if an item has been started and the argument 'next_byte' is smaller than 256 that argument will be appended as the next byte to the buffer.

Parameters

<i>iobuf</i>	The I/O buffer descriptor
<i>next_byte</i>	The value of the next byte or ≥ 256
<i>increment</i>	The no. of bytes by which to increase the buffer beyond the current point. If there is remaining space for writing, the buffer is extended by less than 'increment'.

Returns

next_byte (modulo 256) if successful, -1 for failure

6.11.3.7 find_io_block()

```
int find_io_block (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

Read byte for byte from the input file specified for the I/O buffer and look for the sync-tag (magic number in little-endian or big-endian byte order. As long as the input is properly synchronized this sync-tag should be found in the first four bytes. Otherwise, input data is skipped until the next sync-tag is found. After the sync tag 10 more bytes (item type, version number, and length field) are read. The type of I/O (raw, buffered, or user-defined) depends on the settings of the I/O block.

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
<i>item_header</i>	An item header structure to be filled in.

Returns

0 (O.k.), -1 (error), or -2 (end-of-file)

6.11.3.8 fltp_to_sfloat()

```
void fltp_to_sfloat (
    const float * fnum,
    uint16_t * snum )
```

Both input and output come as pointers to avoid extra conversions.

Parameters

<i>fnum</i>	Pointer to the number to be converted.
<i>snum</i>	Pointer for the resulting representation, as stored in an unsigned 16-bit integer (1 bit sign, 5 bits exponent, 10 bits mantissa).

Here is the caller graph for this function:

**6.11.3.9 free_io_buffer()**

```
void free_io_buffer (
    IO_BUFFER * iobuf )
```

Free an I/O buffer that has been allocated at run-time (e.g. by a call to `allocate_io_buf()`).

Parameters

<i>iobuf</i>	The buffer descriptor to be de-allocated.
--------------	---

Returns

(none)

6.11.3.10 get_count()

```
uintmax_t get_count (
    IO_BUFFER * iobuf )
```

Get an unsigned integer of unspecified length from an I/O buffer where it is encoded in a way similar to the UTF-8 character encoding. Even though the scheme in principle allows for arbitrary length data, the current implementation is limited for data of up to 64 bits. On systems with `uintmax_t` shorter than 64 bits, the result could be clipped unnoticed. It could also be clipped unnoticed in the application calling this function.

6.11.3.11 get_count16()

```
uint16_t get_count16 (
    IO_BUFFER * iobuf )
```

Get an unsigned 16 bit integer of unspecified length from an I/O buffer where it is encoded in a way similar to the UTF-8 character encoding. This is a shorter version of `get_count`, for efficiency reasons.

6.11.3.12 get_count32()

```
uint32_t get_count32 (
    IO_BUFFER * iobuf )
```

Get an unsigned 32 bit integer of unspecified length from an I/O buffer where it is encoded in a way similar to the UTF-8 character encoding. This is a shorter version of `get_count`, for efficiency reasons.

6.11.3.13 get_double()

```
double get_double (
    IO_BUFFER * iobuf )
```

Get a double-precision floating point number (as written by `put_double`) from the I/O buffer. The current implementation is only for machines using IEEE format internally.

Parameters

<i>iobuf</i>	– The I/O buffer descriptor;
--------------	------------------------------

Returns

The floating point number.

6.11.3.14 get_int32()

```
int32_t get_int32 (
    IO_BUFFER * iobuf )
```

Read a four byte integer with little-endian or big-endian byte order from memory. Should be machine independent (see `put_short()`).

6.11.3.15 get_item_begin()

```
int get_item_begin (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

Reads the header of an item.

Reads the header of an item. If a specific item type is requested but a different type is found and the length of that item is known, the item is skipped.

Parameters

<i>iobuf</i>	The input buffer descriptor.
<i>item_header</i>	The item header descriptor.

Returns

0 (O.k.), -1 (error), -2 (end-of-buffer) or -3 (wrong item type).

6.11.3.16 get_item_end()

```
int get_item_end (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

Finish reading an item. The pointer in the I/O buffer is at the end of the item after this call, if succesful.

Parameters

<i>iobuf</i>	I/O buffer descriptor.
<i>item_header</i>	Header of item last read.

Returns

0 (ok), -1 (error)

6.11.3.17 get_long()

```
long get_long (
    IO_BUFFER * iobuf )
```

Read a four byte integer with little-endian or big-endian byte order from memory. Should be machine independent (see [put_short\(\)](#)).

6.11.3.18 get_long_string()

```
int get_long_string (
    char * s,
    int nmax,
    IO_BUFFER * iobuf )
```

Get a long string of ASCII characters with leading count of bytes from an I/O buffer. Strings can be up to $2^{31}-1$ bytes long (assuming you have so much memory).

To work properly with strings longer than 32k, a machine with `sizeof(int) > 2` is actually required.

NOTE: the `nmax` count does account also for the trailing zero byte which will be appended.

6.11.3.19 get_real()

```
double get_real (
    IO_BUFFER * iobuf )
```


Parameters

<i>iobuf</i>	The I/O buffer descriptor;
--------------	----------------------------

Returns

The floating point number.

6.11.3.20 get_scount()

```
intmax_t get_scount (
    IO_BUFFER * iobuf )
```

Get a signed integer of unspecified length from an I/O buffer where it is encoded in a way similar to the UTF-8 character encoding. Even though the scheme in principle allows for arbitrary length data, the current implementation is limited for data of up to 64 bits. On systems with `intmax_t` shorter than 64 bits, the result could be clipped unnoticed.

6.11.3.21 get_short()

```
int get_short (
    IO_BUFFER * iobuf )
```

Get a two-byte integer with least significant byte first. Should be machine-independent (see [put_short\(\)](#)).

6.11.3.22 get_string()

```
int get_string (
    char * s,
    int nmax,
    IO_BUFFER * iobuf )
```

Get a string of ASCII characters with leading count of bytes (stored with 16 bits) from an I/O buffer.

NOTE: the `nmax` count does now account for the trailing zero byte which will be appended. This was different in an earlier version of this function where one additional byte had to be available for the trailing zero byte.

6.11.3.23 get_uint16()

```
uint16_t get_uint16 (
    IO_BUFFER * iobuf )
```

Get one unsigned short (16-bit unsigned int) from an I/O buffer. The function should be used where sign propagation is of concern.

Parameters

<i>iobuf</i>	The output buffer descriptor.
--------------	-------------------------------

Returns

The value obtained from the I/O buffer.

6.11.3.24 get_uint32()

```
uint32_t get_uint32 (
    IO_BUFFER * iobuf )
```

Read a four byte integer with little-endian or big-endian byte order from memory. Should be machine independent (see [put_short\(\)](#)).

6.11.3.25 get_var_string()

```
int get_var_string (
    char * s,
    int nmax,
    IO_BUFFER * iobuf )
```

Get a string of ASCII characters with leading count of bytes (stored with variable length) from an I/O buffer.

NOTE: the nmax count does also account for the trailing zero byte which will be appended.

6.11.3.26 get_vector_of_byte()

```
void get_vector_of_byte (
    BYTE * vec,
    int num,
    IO_BUFFER * iobuf )
```

Parameters

<i>vec</i>	– Byte data vector.
<i>num</i>	– Number of bytes to get.
<i>iobuf</i>	– I/O buffer descriptor.

Returns

(none)

6.11.3.27 get_vector_of_uint16()

```
void get_vector_of_uint16 (
    uint16_t * uval,
    int num,
    IO_BUFFER * iobuf )
```

Get a vector of unsigned shorts from an I/O buffer with least significant byte first. The values are in the range 0 to 65535. The function should be used where sign propagation is of concern.

Parameters

<i>uval</i>	The vector where the values should be loaded.
<i>num</i>	The number of elements to load.
<i>iobuf</i>	The output buffer descriptor.

Returns

(none)

6.11.3.28 get_vector_of_uint16_scount_differential()

```
void get_vector_of_uint16_scount_differential (
    uint16_t * vec,
    int num,
    IO_BUFFER * iobuf )
```

For optimization reasons it is assumed that the data has been written in a consistent way and is complete. Only minimal tests for remaining data in the buffer are made - while the slower generic version would check for each extracted number.

6.11.3.29 get_vector_of_uint32_scount_differential()

```
void get_vector_of_uint32_scount_differential (
    uint32_t * vec,
    int num,
    IO_BUFFER * iobuf )
```

For optimization reasons it is assumed that the data has been written in a consistent way and is complete. Only minimal tests for remaining data in the buffer are made - while the slower generic version would check for each extracted number.

6.11.3.30 list_io_blocks()

```
int list_io_blocks (
    IO_BUFFER * iobuf,
    int verbosity )
```

List type, version, ident, and length) of the top item of all I/O blocks in input file onto standard output.

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
<i>verbosity</i>	Try showing type name at >=1, description at >=2.

Returns

0 (O.k.), -1 (error)

6.11.3.31 list_sub_items()

```
int list_sub_items (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header,
    int maxlevel,
    int verbosity )
```

Display the contents (item types, versions, idents and lengths) of sub-items on standard output.

Parameters

<i>iobuf</i>	I/O buffer descriptor.
<i>item_header</i>	Header of the item from which to show contents.
<i>maxlevel</i>	The maximum nesting depth to show contents (counted from the top-level item on).
<i>verbosity</i>	Try showing type name at ≥ 1 , description at ≥ 2 .

Returns

0 (ok), -1 (error)

6.11.3.32 next_subitem_ident()

```
long next_subitem_ident (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	The input buffer descriptor.
--------------	------------------------------

Returns

≥ 0 (O.k.), -1 (error), -2 (end-of-buffer).

6.11.3.33 next_subitem_length()

```
long next_subitem_length (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	The input buffer descriptor.
--------------	------------------------------

Returns

≥ 0 (O.k.), -1 (error), -2 (end-of-buffer).

6.11.3.34 next_subitem_type()

```
int next_subitem_type (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	The input buffer descriptor.
--------------	------------------------------

Returns

≥ 0 (O.k.), -1 (error), -2 (end-of-buffer).

6.11.3.35 put_count()

```
void put_count (
    uintmax_t n,
    IO_BUFFER * iobuf )
```

Put an unsigned integer of unspecified length in a way similar to the UTF-8 character encoding to an I/O buffer. The byte order resulting in the buffer is independent of the host byte order or the byte order in action for the I/O buffer, starting with as many leading bits in the first byte as extension bytes needed after the first byte. While the scheme in principle allows for values of arbitrary length, the implementation is limited to 64 bits.

Parameters

<i>n</i>	The number to be saved. Even on systems with 64-bit integers, this must not exceed $2^{32}-1$ with the current implementation.
<i>iobuf</i>	The output buffer descriptor.

Returns

(none)

6.11.3.36 put_count16()

```
void put_count16 (
    uint16_t n,
    IO_BUFFER * iobuf )
```

Returns

(none)

6.11.3.37 put_count32()

```
void put_count32 (
    uint32_t n,
    IO_BUFFER * iobuf )
```

Returns

(none)

6.11.3.38 put_double()

```
void put_double (
    double dnum,
    IO_BUFFER * iobuf )
```

Put a 'double' (floating point) number in a specific but (almost) machine-independent format into an I/O buffer. This implementation requires the machine to use IEEE double-precision floating point numbers. Only byte order conversion is done.

Parameters

<i>dnum</i>	The number to be put into the I/O buffer.
<i>iobuf</i>	The I/O buffer descriptor.

Returns

(none)

6.11.3.39 put_int32()

```
void put_int32 (
    int32_t num,
    IO_BUFFER * iobuf )
```

Write a four-byte integer with least significant bytes first. Should be machine independent (see [put_short\(\)](#)).

6.11.3.40 put_item_begin()

```
int put_item_begin (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

When putting another item to the output buffer which may be either a top item or a sub-item, [put_item_begin\(\)](#) initializes the buffer (for a top item) and puts the item header on the buffer.

Parameters

<i>iobuf</i>	The output buffer descriptor.
<i>item_header</i>	The item header descriptor.

Returns

0 (O.k.) or -1 (error)

6.11.3.41 put_item_begin_with_flags()

```
int put_item_begin_with_flags (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header,
    int user_flag,
    int extended )
```

This is identical to [put_item_begin\(\)](#) except for taking a third and fourth argument, a user flag to be included in the header data, and a flag indicating that the header extension should be used. In [put_item_begin\(\)](#) these flags are forced to 0 (false) for backwards compatibility.

Parameters

<i>iobuf</i>	The output buffer descriptor.
<i>item_header</i>	The item header descriptor.
<i>flag</i>	The user flag (0 or 1).

Returns

0 (O.k.) or -1 (error)

6.11.3.42 put_item_end()

```
int put_item_end (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

When finished with putting an item to the output buffer, check for errors and do housekeeping.

Parameters

<i>iobuf</i>	The output buffer descriptor.
<i>item_header</i>	The item header descriptor.

Returns

0 (O.k.) or -1 (error)

6.11.3.43 put_long()

```
void put_long (
    long num,
    IO_BUFFER * iobuf )
```

Write a four-byte integer with least significant bytes first. Should be machine independent (see [put_short\(\)](#)).

6.11.3.44 `put_long_string()`

```
int put_long_string (
    const char * s,
    IO_BUFFER * iobuf )
```

Put a long string of ASCII characters with leading count of bytes into an I/O buffer. This is expected to work properly for strings of more than 32k only on machines with `sizeof(int) > 2` because 16-bit machines may not be able to represent lengths of long strings (as obtained with `strlen`).

Parameters

<i>s</i>	The null-terminated ASCII string.
<i>iobuf</i>	The I/O buffer descriptor.

Returns

Length of string

6.11.3.45 `put_real()`

```
void put_real (
    double dnum,
    IO_BUFFER * iobuf )
```

Put a 'double' (floating point) number in a specific but (almost) machine-independent format into an I/O buffer. Not the full precision of a 'double' is saved but a 32 bit IEEE floating point number is written (with the same byte ordering as long integers). On machines with other floating point format than IEEE the input number is converted to a IEEE number first. An optimized (machine- specific) version should compute the output data by shift and add operations rather than by `log()`, divide, and multiply operations on such non-IEEE-format machines (implemented for VAX only).

Parameters

<i>dnum</i>	The number to be put into the I/O buffer.
<i>iobuf</i>	The I/O buffer descriptor.

Returns

(none)

6.11.3.46 `put_scount()`

```
void put_scount (
    intmax_t n,
    IO_BUFFER * iobuf )
```


Put a signed integer of unspecified length in a way similar to the UTF-8 character encoding to an I/O buffer. The byte order resulting in the buffer is independent of the host byte order or the byte order in action for the I/O buffer, starting with as many leading bits in the first byte as extension bytes needed after the first byte. While the scheme in principle allows for values of arbitrary length, the implementation is limited to 32 bits. To allow an efficient representation of negative numbers, the sign bit is stored in the least significant bit. Portability of data across machines with different `intmax_t` sizes and the need to represent also the most negative number ($-(2^{31})$, $-(2^{63})$, or $-(2^{127})$, depending on CPU type and compiler) is achieved by putting the number's modulus minus 1 into the higher bits.

Parameters

<i>n</i>	The number to be saved. It can be in the range from $-(2^{63})$ to $2^{63}-1$ on systems with 64 bit integers (intrinsic or through the compiler) and from $-(2^{31})$ to $2^{31}-1$ on pure 32 bit systems.
<i>iobuf</i>	The output buffer descriptor.

Returns

(none)

6.11.3.47 put_scount16()

```
void put_scount16 (
    int16_t n,
    IO_BUFFER * iobuf )
```

Apart from efficiency, the data can be read with identical results through `get_scount16` or `get_scount`.

Returns

(none)

6.11.3.48 put_scount32()

```
void put_scount32 (
    int32_t n,
    IO_BUFFER * iobuf )
```

Apart from efficiency, the data can be read with identical results through `get_scount32` or `get_scount`.

Returns

(none)

6.11.3.49 put_short()

```
void put_short (
    int num,
    IO_BUFFER * iobuf )
```

Put a two-byte integer on an I/O buffer with least significant byte first. Should be machine independent as long as 'short' and 'unsigned short' are 16-bit integers, the two's complement is used for negative numbers, and the '>>' operator does a logical shift with unsigned short. Although the 'num' argument is a 4-byte integer on most machines, the value should be in the range -32768 to 32767.

Parameters

<i>num</i>	The number to be saved. Should fit into a short integer and will be truncated otherwise.
<i>iobuf</i>	The output buffer descriptor.

Returns

(none)

6.11.3.50 put_string()

```
int put_string (
    const char * s,
    IO_BUFFER * iobuf )
```

Put a string of ASCII characters with leading count of bytes (stored with 16 bits) into an I/O buffer.

Parameters

<i>s</i>	The null-terminated ASCII string.
<i>iobuf</i>	The I/O buffer descriptor.

Returns

Length of string

6.11.3.51 put_uint32()

```
void put_uint32 (
    uint32_t num,
    IO_BUFFER * iobuf )
```

Write a four-byte integer with least significant bytes first. Should be machine independent (see [put_short\(\)](#)).

6.11.3.52 put_var_string()

```
int put_var_string (
    const char * s,
    IO_BUFFER * iobuf )
```

Put a string of ASCII characters with leading count of bytes (stored with variable length) into an I/O buffer. Note that storing strings of 32k or more length will not work on systems with `sizeof(int)==2`.

Parameters

<i>s</i>	The null-terminated ASCII string.
<i>iobuf</i>	The I/O buffer descriptor.

Returns

Length of string

6.11.3.53 put_vector_of_byte()

```
void put_vector_of_byte (
    const BYTE * vec,
    int num,
    IO_BUFFER * iobuf )
```

Parameters

<i>vec</i>	Byte data vector.
<i>num</i>	Number of bytes to be put.
<i>iobuf</i>	I/O buffer descriptor.

Returns

(none)

6.11.3.54 put_vector_of_double()

```
void put_vector_of_double (
    const double * dvec,
    int num,
    IO_BUFFER * iobuf )
```

Put a vector of 'double' floating point numbers as IEEE 'double' numbers into an I/O buffer.

6.11.3.55 put_vector_of_int()

```
void put_vector_of_int (
    const int * vec,
    int num,
    IO_BUFFER * iobuf )
```

Put a vector of integers (with actual values in the range -32768 to 32767) into an I/O buffer. This may be relaxed by a more efficient but machine-dependent version later.

6.11.3.56 put_vector_of_short()

```
void put_vector_of_short (
    const short * vec,
    int num,
    IO_BUFFER * iobuf )
```

Put a vector of 2-byte integers on an I/O buffer. This may be relaxed by a more efficient but machine-dependent version later. May be called by a number of elements equal to 0. In this case, nothing is done.

6.11.3.57 put_vector_of_uint16()

```
void put_vector_of_uint16 (
    const uint16_t * uval,
    int num,
    IO_BUFFER * iobuf )
```

Put a vector of unsigned shorts into an I/O buffer with least significant byte first. The values are in the range 0 to 65535. The function should be used where sign propagation is of concern.

Parameters

<i>uval</i>	The vector of values to be saved.
<i>num</i>	The number of elements to save.
<i>iobuf</i>	The output buffer descriptor.

Returns

(none)

6.11.3.58 read_io_block()

```
int read_io_block (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

This function is called for reading data after an I/O data block has been found (with `find_io_block`) on input. The type of I/O (raw, buffered, or user-defined) depends on the settings of the I/O block.

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
<i>item_header</i>	The item header descriptor.

Returns

0 (O.k.), -1 (error), -2 (end-of-file), -3 (block skipped because it is too large)

6.11.3.59 remove_item()

```
int remove_item (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

If writing an item has already started and then some condition was found to remove the item again, this is the function for it. The item to be removed should be the last one written, since anything following it will be forgotten too.

Parameters

<i>iobuf</i>	I/O buffer descriptor.
<i>item_header</i>	Header of item to be removed.

Returns

0 (ok), -1 (error)

6.11.3.60 reset_io_block()

```
int reset_io_block (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
--------------	----------------------------

Returns

0 (O.k.), -1 (error)

6.11.3.61 rewind_item()

```
int rewind_item (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

When reading from an I/O buffer, go back to the beginning of the data area of an item. This is typically used when searching for different types of sub-blocks but processing should not depend on the relative order of them.

Parameters

<i>iobuf</i>	I/O buffer descriptor.
<i>item_header</i>	Header of item last read.

Returns

0 (ok), -1 (error)

6.11.3.62 search_sub_item()

```
int search_sub_item (
    IO_BUFFER * iobuf,
```

```
IO_ITEM_HEADER * item_header,  
IO_ITEM_HEADER * sub_item_header )
```

Search for an item of a specified type, starting at the current position in the I/O buffer. After successful action the buffer data pointer points to the beginning of the header of the first item of that type. If no such item is found, it points right after the end of the item of the next higher level.

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
<i>item_header</i>	The header of the item within which we search.
<i>sub_item_header</i>	To be filled with what we found.

Returns

0 (O.k., sub-item was found), -1 (error), -2 (no such sub-item), -3 (cannot skip sub-items),

6.11.3.63 set_eventio_registry_hook()

```
void set_eventio_registry_hook (  
    EVREGSEARCH fptr )
```

6.11.3.64 skip_io_block()

```
int skip_io_block (  
    IO_BUFFER * iobuf,  
    IO_ITEM_HEADER * item_header )
```

Skip the data of an I/O block from the input (after the block's header was read). This is the alternative to [read_io_block\(\)](#) after having found an I/O block with [find_io_block](#) but realizing that this is a type of block you don't know how to read or simply not interested in. The type of I/O (raw, buffered, or user-defined) depends on the settings of the I/O block.

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
<i>item_header</i>	The item header descriptor.

Returns

0 (O.k.), -1 (error) or -2 (end-of-file)

6.11.3.65 skip_subitem()

```
int skip_subitem (  
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor.
--------------	------------------------

Returns

0 (ok), -1 (error)

6.11.3.66 unget_item()

```
int unget_item (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

When reading from an I/O buffer, go back to the beginning of an item (more precisely: its header) currently being read.

Parameters

<i>iobuf</i>	I/O buffer descriptor.
<i>item_header</i>	Header of item last read.

Returns

0 (ok), -1 (error)

6.11.3.67 unput_item()

```
int unput_item (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * item_header )
```

When writing to an I/O buffer, revert anything yet written at the present level. If the buffer was extended, the last length is kept.

Parameters

<i>iobuf</i>	I/O buffer descriptor.
<i>item_header</i>	Header of item last read.

Returns

0 (ok), -1 (error)

6.11.3.68 write_io_block()

```
int write_io_block (
    IO_BUFFER * iobuf )
```

The complete I/O block is written to the output destination, which can be raw I/O (through write), buffered I/O (through fwrite) or user-defined I/O (through a user funtion). All items must have been closed before.

Parameters

<i>iobuf</i>	The I/O buffer descriptor.
--------------	----------------------------

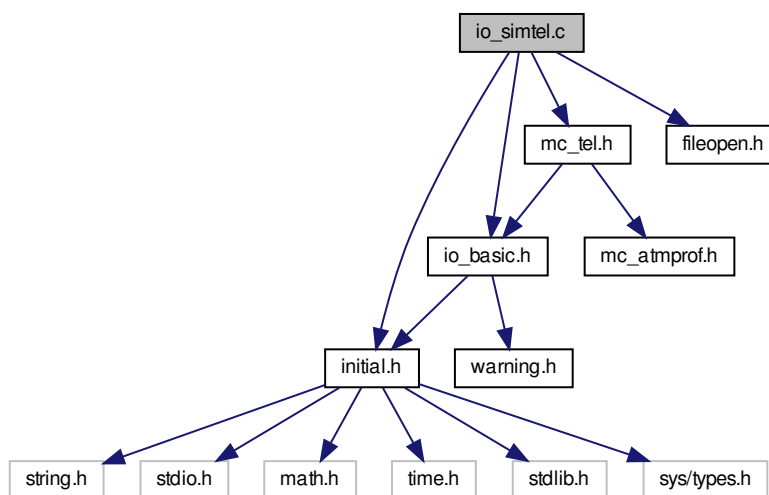
Returns

0 (O.k.), -1 (error), -2 (item has no data)

6.12 io_simtel.c File Reference

Write and read CORSIKA blocks and simulated Cherenkov photon bunches.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "fileopen.h"
Include dependency graph for io_simtel.c:
```



Functions

- int [begin_read_tel_array](#) (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int *array)
Begin reading data for one array of telescopes/detectors.
- int [begin_write_tel_array](#) (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int array)

- Begin writing data for one array of telescopes/detectors.*

 - int `clear_shower_extra_parameters` (`ShowerExtraParam *ep`)

Similar to `init_shower_extra_parameters()` but without any attempts to re-allocate or resize buffers.
- int `end_read_tel_array` (`IO_BUFFER *iobuf`, `IO_ITEM_HEADER *ih`)

End reading data for one array of telescopes/detectors.
- int `end_write_tel_array` (`IO_BUFFER *iobuf`, `IO_ITEM_HEADER *ih`)

End writing data for one array of telescopes/detectors.
- `ShowerExtraParam *get_shower_extra_parameters` ()
- int `init_shower_extra_parameters` (`ShowerExtraParam *ep`, `size_t ni_max`, `size_t nf_max`)

Initialize, resize, clear shower extra parameters.
- int `print_atmprof` (`IO_BUFFER *iobuf`)

Print the atmospheric profile table as used in CORSIKA.
- int `print_camera_layout` (`IO_BUFFER *iobuf`)

Print the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.
- int `print_photo_electrons` (`IO_BUFFER *iobuf`)

List the the photoelectrons registered in a Cherenkov telescope camera.
- int `print_shower_extra_parameters` (`IO_BUFFER *iobuf`)
- int `print_shower_longitudinal` (`IO_BUFFER *iobuf`)

Print CORSIKA shower longitudinal distributions.
- int `print_tel_block` (`IO_BUFFER *iobuf`)

Print a CORSIKA header/trailer block of any type (see `mc_tel.h`)
- int `print_tel_offset` (`IO_BUFFER *iobuf`)

Print offsets and weights of randomly scattered arrays with respect to shower core.
- int `print_tel_photons` (`IO_BUFFER *iobuf`)
- int `print_tel_pos` (`IO_BUFFER *iobuf`)

Print positions of telescopes/detectors within a system or array.
- int `read_atmprof` (`IO_BUFFER *iobuf`, `AtmProf *atmprof`)

Read the atmospheric profile table as used in CORSIKA.
- int `read_camera_layout` (`IO_BUFFER *iobuf`, `int max_pixels`, `int *itel`, `int *type`, `int *pixels`, `double *xp`, `double *yp`)

Read the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.
- int `read_input_lines` (`IO_BUFFER *iobuf`, `struct linked_string *list`)

Read a block with several character strings (normally containing the text of the CORSIKA inputs file) into a linked list.
- int `read_photo_electrons` (`IO_BUFFER *iobuf`, `int max_pixels`, `int max_pe`, `int *array`, `int *tel`, `int *npe`, `int *pixels`, `int *flags`, `int *pe_counts`, `int *tstart`, `double *t`, `double *a`, `int *photon_counts`)

Read the photoelectrons registered in a Cherenkov telescope camera.
- int `read_shower_extra_parameters` (`IO_BUFFER *iobuf`, `ShowerExtraParam *ep`)
- int `read_shower_longitudinal` (`IO_BUFFER *iobuf`, `int *event`, `int *type`, `double *data`, `int ndim`, `int *np`, `int *nthick`, `double *thickstep`, `int max_np`)

Read CORSIKA shower longitudinal distributions.
- int `read_tel_array_end` (`IO_BUFFER *iobuf`, `IO_ITEM_HEADER *ih`, `int *array`)

End reading data for one array of telescopes/detectors.
- int `read_tel_array_head` (`IO_BUFFER *iobuf`, `IO_ITEM_HEADER *ih`, `int *array`)

Begin reading data for one array of telescopes/detectors.
- int `read_tel_block` (`IO_BUFFER *iobuf`, `int type`, `real *data`, `int maxlen`)

Read a CORSIKA header/trailer block of given type (see `mc_tel.h`)
- int `read_tel_offset` (`IO_BUFFER *iobuf`, `int max_array`, `int *narray`, `double *toff`, `double *xoff`, `double *yoff`)

Read offsets of randomly scattered arrays with respect to shower core.
- int `read_tel_offset_w` (`IO_BUFFER *iobuf`, `int max_array`, `int *narray`, `double *toff`, `double *xoff`, `double *yoff`, `double *weight`)

Read offsets and weights of randomly scattered arrays with respect to shower core.

- int [read_tel_photons](#) ([IO_BUFFER](#) *iobuf, int max_bunches, int *array, int *tel, double *photons, struct [bunch](#) *bunches, int *nbunches)
Read bunches of Cherenkov photons for one telescope/detector.
- int [read_tel_pos](#) ([IO_BUFFER](#) *iobuf, int max_tel, int *ntel, double *x, double *y, double *z, double *r)
Read positions of telescopes/detectors within a system or array.
- int [write_atmprof](#) ([IO_BUFFER](#) *iobuf, [AtmProf](#) *atmprof)
Write the atmospheric profile table as used in CORSIKA with ATMEXT option and set up with 'ATMOSPHERE <n> <ref>' or 'IACT ATMOFILE <name>' data cards.
- int [write_camera_layout](#) ([IO_BUFFER](#) *iobuf, int itel, int type, int pixels, double *xp, double *yp)
Write the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.
- int [write_input_lines](#) ([IO_BUFFER](#) *iobuf, struct [linked_string](#) *list)
Write a linked list of character strings (normally containing the text of the CORSIKA inputs file) as a dedicated block.
- int [write_photo_electrons](#) ([IO_BUFFER](#) *iobuf, int array, int tel, int npe, int flags, int pixels, int *pe_counts, int *tstart, double *t, double *a, int *photon_counts)
Write the photo-electrons registered in a Cherenkov telescope camera.
- int [write_shower_extra_parameters](#) ([IO_BUFFER](#) *iobuf, [ShowerExtraParam](#) *ep)
• int [write_shower_longitudinal](#) ([IO_BUFFER](#) *iobuf, int event, int type, double *data, int ndim, int np, int nthick, double thickstep)
Write CORSIKA shower longitudinal distributions.
- int [write_tel_array_end](#) ([IO_BUFFER](#) *iobuf, [IO_ITEM_HEADER](#) *ih, int array)
End writing data for one array of telescopes/detectors.
- int [write_tel_array_head](#) ([IO_BUFFER](#) *iobuf, [IO_ITEM_HEADER](#) *ih, int array)
Begin writing data for one array of telescopes/detectors.
- int [write_tel_block](#) ([IO_BUFFER](#) *iobuf, int type, int num, real *data, int len)
Write a CORSIKA block as given type number (see [mc_tel.h](#)).
- int [write_tel_compact_photons](#) ([IO_BUFFER](#) *iobuf, int array, int tel, double photons, struct [compact_bunch](#) *cbunches, int nbunches, int ext_bunches, char *ext_fname)
Write all the photon bunches for one telescope to an I/O buffer.
- int [write_tel_offset](#) ([IO_BUFFER](#) *iobuf, int narray, double toff, double *xoff, double *yoff)
Write offsets of randomly scattered arrays with respect to shower core.
- int [write_tel_offset_w](#) ([IO_BUFFER](#) *iobuf, int narray, double toff, double *xoff, double *yoff, double *weight)
Write offsets and weights of randomly scattered arrays with respect to shower core.
- int [write_tel_photons](#) ([IO_BUFFER](#) *iobuf, int array, int tel, double photons, struct [bunch](#) *bunches, int nbunches, int ext_bunches, char *ext_fname)
Write all the photon bunches for one telescope to an I/O buffer.
- int [write_tel_pos](#) ([IO_BUFFER](#) *iobuf, int ntel, double *x, double *y, double *z, double *r)
Write positions of telescopes/detectors within a system or array.

Variables

- static int [max_print](#) = 0
Print bunches of Cherenkov photons for one telescope/detector.
- static [ShowerExtraParam](#) [private_shower_extra_parameters](#)
There is one global (more precisely: static) block of extra shower parameters as, for example, used in the CORSIKA IACT interface.

6.12.1 Detailed Description

This file provides functions for writing and reading of CORSIKA header and trailer blocks, positions of telescopes/detectors, lists of simulated Cherenkov photon bunches before any detector simulation for the telescopes as well as of photoelectrons after absorption, telescope ray-tracing and quantum efficiency applied.

Author

Konrad Bernloehr

Date

1997 to 2019

CVS \$Date: 2019/08/07 11:34:07 \$

Version

CVS \$Revision: 1.46 \$

6.12.2 Function Documentation

6.12.2.1 begin_read_tel_array()

```
int begin_read_tel_array (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int * array )
```

Note: this function does not finish reading from the I/O block but after reading of the photons a call to [end_read_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	– I/O buffer descriptor
<i>ih</i>	– I/O item header (for item opened here)
<i>array</i>	– Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.2 begin_write_tel_array()

```
int begin_write_tel_array (
    IO_BUFFER * iobuf,
```

```
IO_ITEM_HEADER * ih,  
int array )
```

Note: this function does not finish writing to the I/O block but after writing of the photons a call to [end_write_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (for item opened here)
<i>array</i>	Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.3 clear_shower_extra_parameters()

```
int clear_shower_extra_parameters (  
    ShowerExtraParam * ep )
```

Just clear contents.

Parameters

<i>ep</i>	Pointer to parameter block. A NULL value indicates that the static block is meant.
-----------	--

6.12.2.4 end_read_tel_array()

```
int end_read_tel_array (  
    IO_BUFFER * iobuf,  
    IO_ITEM_HEADER * ih )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.5 end_write_tel_array()

```
int end_write_tel_array (  
    IO_BUFFER * iobuf,  
    IO_ITEM_HEADER * ih )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.6 init_shower_extra_parameters()

```
int init_shower_extra_parameters (
    ShowerExtraParam * ep,
    size_t ni_max,
    size_t nf_max )
```

Parameters

<i>ep</i>	Pointer to parameter block. A NULL value indicates that the static block is meant.
<i>ni_max</i>	The number of integer parameters to be used.
<i>nf_max</i>	The number of float parameters to be used.

6.12.2.7 print_atmprof()

```
int print_atmprof (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.8 print_camera_layout()

```
int print_camera_layout (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.9 print_photo_electrons()

```
int print_photo_electrons (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.10 print_shower_longitudinal()

```
int print_shower_longitudinal (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.11 print_tel_block()

```
int print_tel_block (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.12 print_tel_offset()

```
int print_tel_offset (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.13 print_tel_pos()

```
int print_tel_pos (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.14 read_atmprof()

```
int read_atmprof (
    IO_BUFFER * iobuf,
    AtmProf * atmprof )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>atmprof</i>	Address of struct with relevant parts of atmospheric profile table

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.15 read_camera_layout()

```
int read_camera_layout (
    IO_BUFFER * iobuf,
```

```

int max_pixels,
int * itel,
int * type,
int * pixels,
double * xp,
double * yp )

```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_pixels</i>	The maximum number of pixels that can be stored in xp, yp.
<i>itel</i>	telescope number
<i>type</i>	camera type (hex/square)
<i>pixels</i>	number of pixels
<i>xp</i>	X positions of pixels
<i>yp</i>	Y position of pixels

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.16 read_input_lines()

```

int read_input_lines (
    IO_BUFFER * iobuf,
    struct linked_string * list )

```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>list</i>	starting point of linked list (on first call this should be a link to an empty list, i.e. the first element has text=NULL and next=NULL; on additional calls the new lines will be appended.)

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.17 read_photo_electrons()

```

int read_photo_electrons (
    IO_BUFFER * iobuf,
    int max_pixels,
    int max_pe,
    int * array,
    int * tel,
    int * npe,
    int * pixels,
    int * flags,

```



```

int * pe_counts,
int * tstart,
double * t,
double * a,
int * photon_counts )

```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_pixels</i>	Maximum number of pixels which can be treated
<i>max_pe</i>	Maximum number of photo-electrons
<i>array</i>	Array number
<i>tel</i>	Telescope number
<i>npe</i>	The total number of photo-electrons read.
<i>pixels</i>	Number of pixels read.
<i>flags</i>	Bit 0: amplitudes available, bit 1: includes NSB p.e.
<i>pe_counts</i>	Numbers of photo-electrons in each pixel
<i>tstart</i>	Offsets in 't' at which data for each pixel starts
<i>t</i>	Time of arrival of photons at the camera.
<i>a</i>	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
<i>photon_counts</i>	Optional number of photons arriving at a pixel.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.18 read_shower_longitudinal()

```

int read_shower_longitudinal (
    IO_BUFFER * iobuf,
    int * event,
    int * type,
    double * data,
    int ndim,
    int * np,
    int * nthick,
    double * thickstep,
    int max_np )

```

See [telling_\(\)](#) in [iact.c](#) for more detailed parameter description.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>event</i>	return event number
<i>type</i>	return 1 = particle numbers, 2 = energy, 3 = energy deposits
<i>data</i>	return set of (usually 9) distributions
<i>ndim</i>	maximum number of entries per distribution
<i>np</i>	return number of distributions (usually 9)
<i>nthick</i>	return number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
<i>thickstep</i>	return step size in g/cm**2
<i>max_np</i>	maximum number of distributions for which we have space.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.19 read_tel_array_end()

```
int read_tel_array_end (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int * array )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.20 read_tel_array_head()

```
int read_tel_array_head (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int * array )
```

Note: this function does not finish reading from the I/O block but after reading of the photons a call to [end_read_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	– I/O buffer descriptor
<i>ih</i>	– I/O item header (for item opened here)
<i>array</i>	– Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.21 read_tel_block()

```
int read_tel_block (
    IO_BUFFER * iobuf,
    int type,
    real * data,
    int maxlen )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>type</i>	block type (see mc_tel.h)
<i>data</i>	area for data to be read
<i>maxlen</i>	maximum number of elements to be read

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.22 read_tel_offset()

```
int read_tel_offset (
    IO_BUFFER * iobuf,
    int max_array,
    int * narray,
    double * toff,
    double * xoff,
    double * yoff )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_array</i>	Maximum number of arrays that can be treated
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.23 read_tel_offset_w()

```
int read_tel_offset_w (
    IO_BUFFER * iobuf,
    int max_array,
    int * narray,
    double * toff,
    double * xoff,
    double * yoff,
    double * weight )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Parameters

<i>max_array</i>	Maximum number of arrays that can be treated
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays
<i>weight</i>	Area weight for uniform or importance sampled core offset. For old version data (uniformly sampled), 0.0 is returned.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.24 read_tel_photons()

```
int read_tel_photons (
    IO_BUFFER * iobuf,
    int max_bunches,
    int * array,
    int * tel,
    double * photons,
    struct bunch * bunches,
    int * nbunches )
```

The data format may be either the more or less compact one.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_bunches</i>	maximum number of bunches that can be treated
<i>array</i>	array number
<i>tel</i>	telescope number
<i>photons</i>	sum of photons (and fractions) in this device
<i>bunches</i>	list of photon bunches
<i>nbunches</i>	number of elements in bunch list

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.25 read_tel_pos()

```
int read_tel_pos (
    IO_BUFFER * iobuf,
    int max_tel,
    int * ntel,
```

```
double * x,
double * y,
double * z,
double * r )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_tel</i>	maximum number of telescopes allowed
<i>ntel</i>	number of telescopes/detectors
<i>x</i>	X positions
<i>y</i>	Y positions
<i>z</i>	Z positions
<i>r</i>	radius of spheres including the whole devices

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.26 write_atmprof()

```
int write_atmprof (
    IO_BUFFER * iobuf,
    AtmProf * atmprof )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>atmprof</i>	Address of struct with relevant parts of atmospheric profile table

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.27 write_camera_layout()

```
int write_camera_layout (
    IO_BUFFER * iobuf,
    int itel,
    int type,
    int pixels,
    double * xp,
    double * yp )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>itel</i>	telescope number

Parameters

<i>type</i>	camera type (hex/square)
<i>pixels</i>	number of pixels
<i>xp</i>	X positions of pixels
<i>yp</i>	Y position of pixels

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.28 write_input_lines()

```
int write_input_lines (
    IO_BUFFER * iobuf,
    struct linked_string * list )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>list</i>	starting point of linked list

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.29 write_photo_electrons()

```
int write_photo_electrons (
    IO_BUFFER * iobuf,
    int array,
    int tel,
    int npe,
    int flags,
    int pixels,
    int * pe_counts,
    int * tstart,
    double * t,
    double * a,
    int * photon_counts )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>array</i>	array number
<i>tel</i>	telescope number
<i>npe</i>	Total number of photo-electrons in the camera.
<i>pixels</i>	No. of pixels to be written

Parameters

<i>flags</i>	Bit 0: save also amplitudes if available, Bit 1: p.e. list includes NSB p.e., bit 2: data also including no. of photons hitting each pixel. bit 3: photons (if any) are in wavelength range 300-550 nm.
<i>pe_counts</i>	Numbers of photo-electrons in each pixel
<i>tstart</i>	Offsets in 't' at which data for each pixel starts
<i>t</i>	Time of arrival of photons at the camera.
<i>a</i>	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
<i>photon_counts</i>	Optional number of photons arriving at a pixel (with flags bit 2 set)

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.30 write_shower_longitudinal()

```
int write_shower_longitudinal (
    IO_BUFFER * iobuf,
    int event,
    int type,
    double * data,
    int ndim,
    int np,
    int nthick,
    double thickstep )
```

See [telling_\(\)](#) in [iact.c](#) for more detailed parameter description.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>event</i>	event number
<i>type</i>	1 = particle numbers, 2 = energy, 3 = energy deposits
<i>data</i>	set of (usually 9) distributions
<i>ndim</i>	maximum number of entries per distribution
<i>np</i>	number of distributions (usually 9)
<i>nthick</i>	number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
<i>thickstep</i>	step size in g/cm**2

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.31 write_tel_array_end()

```
int write_tel_array_end (
    IO_BUFFER * iobuf,
```

```
IO_ITEM_HEADER * ih,  
int array )
```


Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.32 write_tel_array_head()

```
int write_tel_array_head (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int array )
```

Note: this function does not finish writing to the I/O block but after writing of the photons a call to [end_write_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (for item opened here)
<i>array</i>	Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.33 write_tel_block()

```
int write_tel_block (
    IO_BUFFER * iobuf,
    int type,
    int num,
    real * data,
    int len )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>type</i>	block type (see mc_tel.h)
<i>num</i>	Run or event number depending on type
<i>data</i>	Data as passed from CORSIKA
<i>len</i>	Number of elements to be written

Returns

0 (OK), -1, -2, -3 (error, as usual in eventio)

6.12.2.34 write_tel_compact_photons()

```
int write_tel_compact_photons (
    IO_BUFFER * iobuf,
    int array,
    int tel,
    double photons,
    struct compact_bunch * cbunches,
    int nbunches,
    int ext_bunches,
    char * ext_fname )
```

Usually, calls to this function for each telescope in an array should be enclosed within calls to [begin_write_tel_array\(\)](#) and [end_write_tel_array\(\)](#). This routine writes the more compact format (16 bytes per bunch). The more compact format should usually be used to save memory and disk space.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>array</i>	array number
<i>tel</i>	telescope number
<i>photons</i>	sum of photons (and fractions) in this device
<i>cbunches</i>	list of photon bunches
<i>nbunches</i>	number of elements in bunch list
<i>ext_bunches</i>	number of elements in external file
<i>ext_fname</i>	name of external (temporary) file

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.35 write_tel_offset()

```
int write_tel_offset (
    IO_BUFFER * iobuf,
    int narray,
    double toff,
    double * xoff,
    double * yoff )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.36 write_tel_offset_w()

```
int write_tel_offset_w (
    IO_BUFFER * iobuf,
    int narray,
    double toff,
    double * xoff,
    double * yoff,
    double * weight )
```

With respect to the backwards-compatible non-weights version [write_tel_offset\(\)](#), this version adds a weight to each offset position which should be normalized in such a way that with uniform sampling it should be the area over which showers are thrown divided by the number of array in each shower. With importance sampling the same relation should hold on average. So in either case, the average sum of weights for the different offsets in one shower equals just the area over which cores are randomized. This leaves the possibility to change the number of offsets from shower to shower.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays
<i>weight</i>	Area weight for uniform or importance sampled core offset.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.37 write_tel_photons()

```
int write_tel_photons (
    IO_BUFFER * iobuf,
    int array,
    int tel,
    double photons,
    struct bunch * bunches,
    int nbunches,
    int ext_bunches,
    char * ext_fname )
```

Usually, calls to this function for each telescope in an array should be enclosed within calls to [begin_write_tel_array\(\)](#) and [end_write_tel_array\(\)](#). This routine writes the less compact format (32 bytes per bunch).

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>array</i>	array number
<i>tel</i>	telescope number
<i>photons</i>	sum of photons (and fractions) in this device
<i>bunches</i>	list of photon bunches
<i>nbunches</i>	number of elements in bunch list
<i>ext_bunches</i>	number of elements in external file
<i>ext_fname</i>	name of external (temporary) file

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.2.38 write_tel_pos()

```
int write_tel_pos (
    IO_BUFFER * iobuf,
    int ntel,
    double * x,
    double * y,
    double * z,
    double * r )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ntel</i>	number of telescopes/detectors
<i>x</i>	X positions
<i>y</i>	Y positions
<i>z</i>	Z positions
<i>r</i>	radius of spheres including the whole devices

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.3 Variable Documentation**6.12.3.1 max_print**

```
int max_print = 0 [static]
```

The data format may be either the more or less compact one.

Parameters

<code>iobuf</code>	I/O buffer descriptor
--------------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.12.3.2 private_shower_extra_parameters

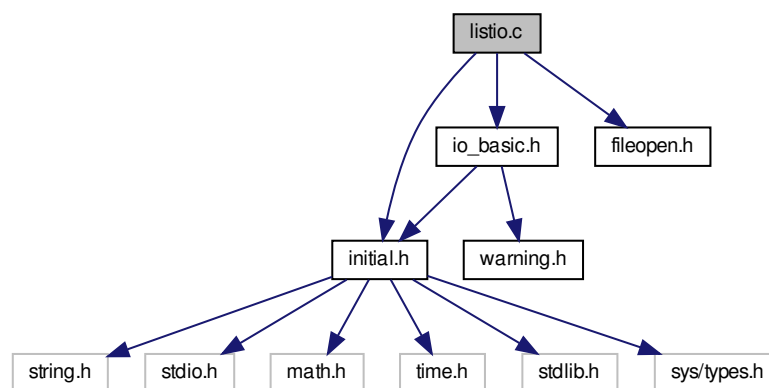
```
ShowExtraParam private_shower_extra_parameters [static]
```

Get a pointer to this block.

6.13 listio.c File Reference

Main function for listing data consisting of eventio blocks.

```
#include "initial.h"
#include "io_basic.h"
#include "fileopen.h"
Include dependency graph for listio.c:
```



Functions

- `int main (int argc, char **argv)`
Main function.

6.13.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2019/07/25 14:26:59 \$

Version

CVS \$Revision: 1.17 \$

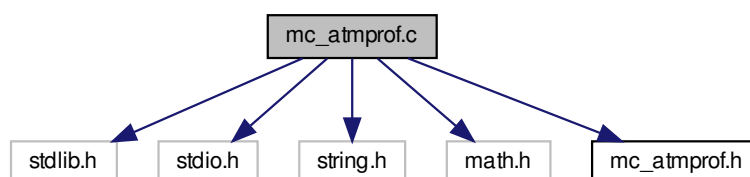
The item type, version, length and ident are displayed. With command line option '-s' all sub-items are shown as well. Input is from standard input by default, output to standard output.

```
Syntax: listio [-s[n]] [-p] [filename]
List structure of eventio data files.
-s : also list contained (sub-) items
-sn: list sub-items up to depth n (n=0,1,...)
-p : show positions of items in the file
If no file name given, standard input is used.
```

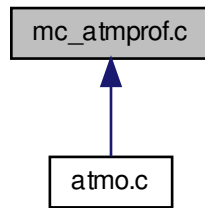
6.14 mc_atmprof.c File Reference

Interface to the atmospheric profile structure.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mc_atmprof.h"
Include dependency graph for mc_atmprof.c:
```



This graph shows which files directly or indirectly include this file:



Functions

- void [atmegs_](#) (int *nlay, double *hlay, double *aatm, double *batm, double *catm, double *datm, double *htoa)
Fill the 5-layer parameters into the common atmospheric profile structure for keeping track of that together with the tabular input.
- void **atmegs_default** (void)
- [AtmProf *](#) [get_common_atmprof](#) (void)
Make this copy of the atmospheric profile available elsewhere.
- double **heighc** (double *thick)
- double **refidc** (double *height)
- double **refim1c** (double *height)
- double **rhofc** (double *height)
C-called functions equivalent to the CORSIKA-built-in functions to evaluate the 5-layer parametrization.
- void [set_common_atmprof](#) ([AtmProf *](#)aprof)
Set the common profile from a separate copy.
- void [show_atmprof](#) ([AtmProf *](#)aprof)
Show a readable version of the tabulated atmospheric profile (basically like in the original tables, except for comments and extra unused columns), plus the 5-layer parametrization, if available.
- double **thickc** (double *height)

Variables

- static [AtmProf](#) [common_atmprof](#)
Keep track of atmospheric profiles loaded from text tables.
- static double **etadsn0** = 0.000283 * 994186.38 / 1222.656

6.14.1 Function Documentation

6.14.1.1 rhofc()

```
double rhofc (
    double * height )
```

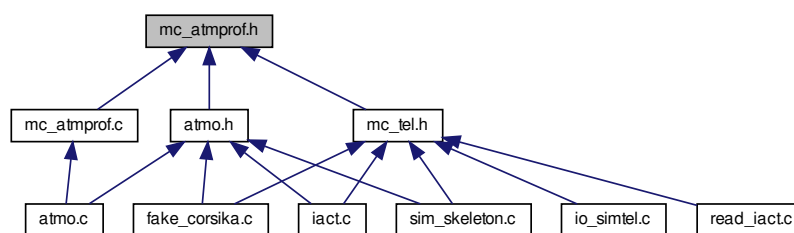
Assumes that these parameters have been set before. Where the numerical table is available it should be used once to initialize the atmospheric profile and then use the corresponding [rhofx_\(\)](#), ... functions for the evaluation instead.

References `atmospheric_profile::batm`, `common_atmprof`, `atmospheric_profile::datm`, and `atmospheric_profile::hlay`.

6.15 mc_atmprof.h File Reference

A data structure shared between [io_simtel.c](#) and [atmo.c](#) - which is used by both `sim_telarray` and the CORSIKA IACT/atmo package.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [atmospheric_profile](#)
Atmospheric profile as stored in atmprof.dat files - the actually used columns only.*

Typedefs

- typedef struct [atmospheric_profile](#) **AtmProf**

Functions

- void [atmegs_](#) (int *nlay, double *hlay, double *aatm, double *batm, double *catm, double *datm, double *htoa)
Fill the 5-layer parameters into the common atmospheric profile structure for keeping track of that together with the tabular input.
- void [atmegs_default](#) (void)
- [AtmProf *](#) [get_common_atmprof](#) (void)
Make this copy of the atmospheric profile available elsewhere.
- double [heighc](#) (double *thick)

- double **refidc** (double *height)
- double **refim1c** (double *height)
- double **rhofc** (double *height)
C-called functions equivalent to the CORSIKA-built-in functions to evaluate the 5-layer parametrization.
- void **set_common_atmprof** (AtmProf *atmprof)
Set the common profile from a separate copy.
- void **show_atmprof** (AtmProf *atmprof)
Show a readable version of the tabulated atmospheric profile (basically like in the original tables, except for comments and extra unused columns), plus the 5-layer parametrization, if available.
- double **thickc** (double *height)

6.15.1 Detailed Description

Filling the structure from text format tables is handled by [atmo.c](#) while EventIO input and output is handled by [io_simtel.c](#). The purpose of the structure is for keeping track of the profile actually used. Evaluating/interpolating it is handled elsewhere. In addition to the tabulated profiles, it can also keep track of the 5-layer parametrization as hard-wired into the CORSIKA EGS part.

Author

Konrad Bernloehr

Date

2019

CVS \$Date: 2019/07/23 16:52:59 \$

Version

CVS \$Revision: 1.2 \$

6.15.2 Function Documentation

6.15.2.1 rhofc()

```
double rhofc (
    double * height )
```

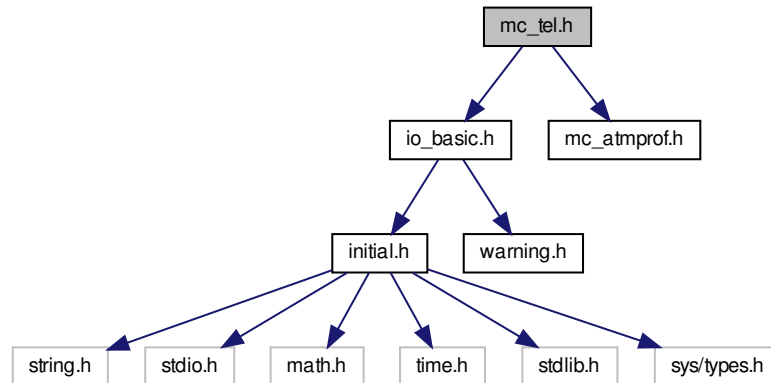
Assumes that these parameters have been set before. Where the numerical table is available it should be used once to initialize the atmospheric profile and then use the corresponding [rhofx_\(\)](#), ... functions for the evaluation instead.

References [atmospheric_profile::batm](#), [common_atmprof](#), [atmospheric_profile::datm](#), and [atmospheric_profile::hlay](#).

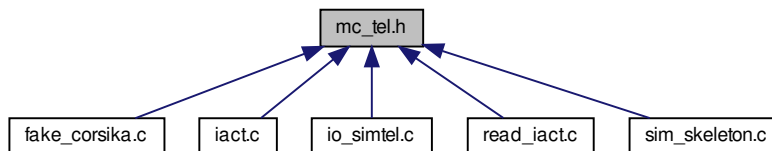
6.16 mc_tel.h File Reference

Definitions and structures for CORSIKA Cherenkov light interface.

```
#include "io_basic.h"
#include "mc_atmprof.h"
Include dependency graph for mc_tel.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [bunch](#)
Photons collected in bunches of identical direction, position, time, and wavelength.
- struct [compact_bunch](#)
The [compact_bunch](#) struct is equivalent to the [bunch](#) struct except that we try to use less memory.
- struct [linked_string](#)
The [linked_string](#) is mainly used to keep CORSIKA input.
- struct [photo_electron](#)
A photo-electron produced by a photon hitting a pixel.
- struct [shower_extra_parameters](#)
Extra shower parameters of unspecified nature.

Macros

- `#define _MC_TEL_LOADED 2`
- `#define IO_TYPE_MC_ATMPROF (IO_TYPE_MC_BASE+16)`
- `#define IO_TYPE_MC_BASE 1200`
- `#define IO_TYPE_MC_EVTE (IO_TYPE_MC_BASE+9)`
- `#define IO_TYPE_MC_EVTH (IO_TYPE_MC_BASE+2)`
- `#define IO_TYPE_MC_EXTRA_PARAM (IO_TYPE_MC_BASE+15)`
- `#define IO_TYPE_MC_INPUTCFG (IO_TYPE_MC_BASE+12)`
- `#define IO_TYPE_MC_LAYOUT (IO_TYPE_MC_BASE+6)`
- `#define IO_TYPE_MC_LONGI (IO_TYPE_MC_BASE+11)`
- `#define IO_TYPE_MC_PE (IO_TYPE_MC_BASE+8)`
- `#define IO_TYPE_MC_PHOTONS (IO_TYPE_MC_BASE+5)`
- `#define IO_TYPE_MC_RUNE (IO_TYPE_MC_BASE+10)`
- `#define IO_TYPE_MC_RUNH (IO_TYPE_MC_BASE+0)`
- `#define IO_TYPE_MC_TELARRAY (IO_TYPE_MC_BASE+4)`
- `#define IO_TYPE_MC_TELARRAY_END (IO_TYPE_MC_BASE+14)`
- `#define IO_TYPE_MC_TELARRAY_HEAD (IO_TYPE_MC_BASE+13)`
- `#define IO_TYPE_MC_TELOFF (IO_TYPE_MC_BASE+3)`
- `#define IO_TYPE_MC_TELPOS (IO_TYPE_MC_BASE+1)`
- `#define IO_TYPE_MC_TRIGTIME (IO_TYPE_MC_BASE+7)`

Typedefs

- `typedef short INT16`
- `typedef int INT32`
- `typedef float real`
- `typedef struct shower_extra_parameters ShowerExtraParam`
- `typedef unsigned short UINT16`
- `typedef unsigned int UINT32`

Functions

- `int begin_read_tel_array (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int *array)`
Begin reading data for one array of telescopes/detectors.
- `int begin_write_tel_array (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int array)`
Begin writing data for one array of telescopes/detectors.
- `int clear_shower_extra_parameters (ShowerExtraParam *ep)`
Similar to `init_shower_extra_parameters()` but without any attempts to re-allocate or resize buffers.
- `int end_read_tel_array (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih)`
End reading data for one array of telescopes/detectors.
- `int end_write_tel_array (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih)`
End writing data for one array of telescopes/detectors.
- `int get_corsika_atm_params (double *hlay, double *aatm, double *batm, double *catm, double *datm)`
- `ShowerExtraParam * get_shower_extra_parameters (void)`
- `int init_shower_extra_parameters (ShowerExtraParam *ep, size_t ni_max, size_t nf_max)`
Initialize, resize, clear shower extra parameters.
- `int print_atmprof (IO_BUFFER *iobuf)`
Print the atmospheric profile table as used in CORSIKA.
- `int print_camera_layout (IO_BUFFER *iobuf)`
Print the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.
- `int print_photo_electrons (IO_BUFFER *iobuf)`

- List the the photoelectrons registered in a Cherenkov telescope camera.*
- int **print_shower_extra_parameters** (IO_BUFFER *iobuf)
- int **print_shower_longitudinal** (IO_BUFFER *iobuf)
- Print CORSIKA shower longitudinal distributions.*
- int **print_tel_block** (IO_BUFFER *iobuf)
- Print a CORSIKA header/trailer block of any type (see [mc_tel.h](#))*
- int **print_tel_offset** (IO_BUFFER *iobuf)
- Print offsets and weights of randomly scattered arrays with respect to shower core.*
- int **print_tel_photons** (IO_BUFFER *iobuf)
- int **print_tel_pos** (IO_BUFFER *iobuf)
- Print positions of telescopes/detectors within a system or array.*
- int **read_atmprof** (IO_BUFFER *iobuf, AtmProf *atmprof)
- Read the atmospheric profile table as used in CORSIKA.*
- int **read_camera_layout** (IO_BUFFER *iobuf, int max_pixels, int *itel, int *type, int *pixels, double *xp, double *yp)
- Read the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.*
- int **read_input_lines** (IO_BUFFER *iobuf, struct linked_string *list)
- Read a block with several character strings (normally containing the text of the CORSIKA inputs file) into a linked list.*
- int **read_photo_electrons** (IO_BUFFER *iobuf, int max_pixel, int max_pe, int *array, int *tel, int *npe, int *pixels, int *flags, int *pe_counts, int *tstart, double *t, double *a, int *photon_counts)
- Read the photoelectrons registered in a Cherenkov telescope camera.*
- int **read_shower_extra_parameters** (IO_BUFFER *iobuf, ShowerExtraParam *ep)
- int **read_shower_longitudinal** (IO_BUFFER *iobuf, int *event, int *type, double *data, int ndim, int *np, int *nthick, double *thickstep, int max_np)
- Read CORSIKA shower longitudinal distributions.*
- int **read_tel_array_end** (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int *array)
- End reading data for one array of telescopes/detectors.*
- int **read_tel_array_head** (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int *array)
- Begin reading data for one array of telescopes/detectors.*
- int **read_tel_block** (IO_BUFFER *iobuf, int type, real *data, int maxlen)
- Read a CORSIKA header/trailer block of given type (see [mc_tel.h](#))*
- int **read_tel_offset** (IO_BUFFER *iobuf, int max_array, int *narray, double *toff, double *xoff, double *yoff)
- Read offsets of randomly scattered arrays with respect to shower core.*
- int **read_tel_offset_w** (IO_BUFFER *iobuf, int max_array, int *narray, double *toff, double *xoff, double *yoff, double *weight)
- Read offsets and weights of randomly scattered arrays with respect to shower core.*
- int **read_tel_photons** (IO_BUFFER *iobuf, int max_bunches, int *array, int *tel, double *photons, struct bunch *bunches, int *nbunches)
- Read bunches of Cherenkov photons for one telescope/detector.*
- int **read_tel_pos** (IO_BUFFER *iobuf, int max_tel, int *ntel, double *x, double *y, double *z, double *r)
- Read positions of telescopes/detectors within a system or array.*
- void **remember_corsika_atm_params** (float *hlay, float *aatm, float *batm, float *catm)
- int **write_atmprof** (IO_BUFFER *iobuf, AtmProf *atmprof)
- Write the atmospheric profile table as used in CORSIKA with ATMEXT option and set up with 'ATMOSPHERE <n> <fref>' or 'IACT ATMOFILE <name>' data cards.*
- int **write_camera_layout** (IO_BUFFER *iobuf, int itel, int type, int pixels, double *xp, double *yp)
- Write the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.*
- int **write_input_lines** (IO_BUFFER *iobuf, struct linked_string *list)
- Write a linked list of character strings (normally containing the text of the CORSIKA inputs file) as a dedicated block.*
- int **write_photo_electrons** (IO_BUFFER *iobuf, int array, int tel, int npe, int pixels, int flags, int *pe_counts, int *tstart, double *t, double *a, int *photon_counts)
- Write the photo-electrons registered in a Cherenkov telescope camera.*

- int **write_shower_extra_parameters** (IO_BUFFER *iobuf, ShowerExtraParam *ep)
- int **write_shower_longitudinal** (IO_BUFFER *iobuf, int event, int type, double *data, int ndim, int np, int nthick, double thickstep)
Write CORSIKA shower longitudinal distributions.
- int **write_tel_array_end** (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int array)
End writing data for one array of telescopes/detectors.
- int **write_tel_array_head** (IO_BUFFER *iobuf, IO_ITEM_HEADER *ih, int array)
Begin writing data for one array of telescopes/detectors.
- int **write_tel_block** (IO_BUFFER *iobuf, int type, int num, real *data, int len)
Write a CORSIKA block as given type number (see mc_tel.h).
- int **write_tel_compact_photons** (IO_BUFFER *iobuf, int array, int tel, double photons, struct compact_bunch *cbunches, int nbunches, int ext_bunches, char *ext_fname)
Write all the photon bunches for one telescope to an I/O buffer.
- int **write_tel_offset** (IO_BUFFER *iobuf, int narray, double toff, double *xoff, double *yoff)
Write offsets of randomly scattered arrays with respect to shower core.
- int **write_tel_offset_w** (IO_BUFFER *iobuf, int narray, double toff, double *xoff, double *yoff, double *weight)
Write offsets and weights of randomly scattered arrays with respect to shower core.
- int **write_tel_photons** (IO_BUFFER *iobuf, int array, int tel, double photons, struct bunch *bunches, int nbunches, int ext_bunches, char *ext_fname)
Write all the photon bunches for one telescope to an I/O buffer.
- int **write_tel_pos** (IO_BUFFER *iobuf, int ntel, double *x, double *y, double *z, double *r)
Write positions of telescopes/detectors within a system or array.

6.16.1 Detailed Description

This file contains definitions of data structures and of function prototypes as needed for the Cherenkov light extraction interfaced to the modified CORSIKA code.

Author

Konrad Bernloehr

Date

1997 to 2019

CVS \$Date: 2019/07/13 18:27:56 \$

Version

CVS \$Revision: 1.20 \$

6.16.2 Function Documentation

6.16.2.1 begin_read_tel_array()

```
int begin_read_tel_array (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int * array )
```

Note: this function does not finish reading from the I/O block but after reading of the photons a call to **end_read_tel_array()** is needed.

Parameters

<i>iobuf</i>	– I/O buffer descriptor
<i>ih</i>	– I/O item header (for item opened here)
<i>array</i>	– Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.2 begin_write_tel_array()

```
int begin_write_tel_array (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int array )
```

Note: this function does not finish writing to the I/O block but after writing of the photons a call to [end_write_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (for item opened here)
<i>array</i>	Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.3 clear_shower_extra_parameters()

```
int clear_shower_extra_parameters (
    ShowerExtraParam * ep )
```

Just clear contents.

Parameters

<i>ep</i>	Pointer to parameter block. A NULL value indicates that the static block is meant.
-----------	--

6.16.2.4 end_read_tel_array()

```
int end_read_tel_array (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.5 end_write_tel_array()

```
int end_write_tel_array (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.6 init_shower_extra_parameters()

```
int init_shower_extra_parameters (
    ShowerExtraParam * ep,
    size_t ni_max,
    size_t nf_max )
```

Parameters

<i>ep</i>	Pointer to parameter block. A NULL value indicates that the static block is meant.
<i>ni_max</i>	The number of integer parameters to be used.
<i>nf_max</i>	The number of float parameters to be used.

6.16.2.7 print_atmprof()

```
int print_atmprof (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.8 print_camera_layout()

```
int print_camera_layout (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.9 print_photo_electrons()

```
int print_photo_electrons (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.10 print_shower_longitudinal()

```
int print_shower_longitudinal (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.11 print_tel_block()

```
int print_tel_block (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.12 print_tel_offset()

```
int print_tel_offset (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.13 print_tel_pos()

```
int print_tel_pos (
    IO_BUFFER * iobuf )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
--------------	-----------------------

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.14 read_atmprof()

```
int read_atmprof (
    IO_BUFFER * iobuf,
    AtmProf * atmprof )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>atmprof</i>	Address of struct with relevant parts of atmospheric profile table

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.15 read_camera_layout()

```
int read_camera_layout (
    IO_BUFFER * iobuf,
    int max_pixels,
    int * itel,
    int * type,
    int * pixels,
    double * xp,
    double * yp )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_pixels</i>	The maximum number of pixels that can be stored in xp, yp.
<i>itel</i>	telescope number
<i>type</i>	camera type (hex/square)
<i>pixels</i>	number of pixels
<i>xp</i>	X positions of pixels
<i>yp</i>	Y position of pixels

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.16 read_input_lines()

```
int read_input_lines (
    IO_BUFFER * iobuf,
    struct linked_string * list )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>list</i>	starting point of linked list (on first call this should be a link to an empty list, i.e. the first element has text=NULL and next=NULL; on additional calls the new lines will be appended.)

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.17 read_photo_electrons()

```
int read_photo_electrons (
    IO_BUFFER * iobuf,
    int max_pixels,
    int max_pe,
    int * array,
    int * tel,
    int * npe,
    int * pixels,
    int * flags,
    int * pe_counts,
    int * tstart,
    double * t,
    double * a,
    int * photon_counts )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_pixels</i>	Maximum number of pixels which can be treated
<i>max_pe</i>	Maximum number of photo-electrons
<i>array</i>	Array number
<i>tel</i>	Telescope number
<i>npe</i>	The total number of photo-electrons read.
<i>pixels</i>	Number of pixels read.
<i>flags</i>	Bit 0: amplitudes available, bit 1: includes NSB p.e.
<i>pe_counts</i>	Numbers of photo-electrons in each pixel
<i>tstart</i>	Offsets in 't' at which data for each pixel starts
<i>t</i>	Time of arrival of photons at the camera.
<i>a</i>	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
<i>photon_counts</i>	Optional number of photons arriving at a pixel.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.18 read_shower_longitudinal()

```
int read_shower_longitudinal (
    IO_BUFFER * iobuf,
    int * event,
    int * type,
    double * data,
```

```

int ndim,
int * np,
int * nthick,
double * thickstep,
int max_np )

```

See [telling_\(\)](#) in [iact.c](#) for more detailed parameter description.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>event</i>	return event number
<i>type</i>	return 1 = particle numbers, 2 = energy, 3 = energy deposits
<i>data</i>	return set of (usually 9) distributions
<i>ndim</i>	maximum number of entries per distribution
<i>np</i>	return number of distributions (usually 9)
<i>nthick</i>	return number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
<i>thickstep</i>	return step size in g/cm**2
<i>max_np</i>	maximum number of distributions for which we have space.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.19 read_tel_array_end()

```

int read_tel_array_end (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int * array )

```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.20 read_tel_array_head()

```

int read_tel_array_head (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int * array )

```

Note: this function does not finish reading from the I/O block but after reading of the photons a call to [end_read_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	– I/O buffer descriptor
<i>ih</i>	– I/O item header (for item opened here)
<i>array</i>	– Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.21 read_tel_block()

```
int read_tel_block (
    IO_BUFFER * iobuf,
    int type,
    real * data,
    int maxlen )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>type</i>	block type (see mc_tel.h)
<i>data</i>	area for data to be read
<i>maxlen</i>	maximum number of elements to be read

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.22 read_tel_offset()

```
int read_tel_offset (
    IO_BUFFER * iobuf,
    int max_array,
    int * narray,
    double * toff,
    double * xoff,
    double * yoff )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_array</i>	Maximum number of arrays that can be treated
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.23 read_tel_offset_w()

```
int read_tel_offset_w (
    IO_BUFFER * iobuf,
    int max_array,
    int * narray,
    double * toff,
    double * xoff,
    double * yoff,
    double * weight )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_array</i>	Maximum number of arrays that can be treated
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays
<i>weight</i>	Area weight for uniform or importance sampled core offset. For old version data (uniformly sampled), 0.0 is returned.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.24 read_tel_photons()

```
int read_tel_photons (
    IO_BUFFER * iobuf,
    int max_bunches,
    int * array,
    int * tel,
    double * photons,
    struct bunch * bunches,
    int * nbunches )
```

The data format may be either the more or less compact one.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_bunches</i>	maximum number of bunches that can be treated
<i>array</i>	array number
<i>tel</i>	telescope number
<i>photons</i>	sum of photons (and fractions) in this device
<i>bunches</i>	list of photon bunches
<i>nbunches</i>	number of elements in bunch list

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.25 read_tel_pos()

```
int read_tel_pos (
    IO_BUFFER * iobuf,
    int max_tel,
    int * ntel,
    double * x,
    double * y,
    double * z,
    double * r )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>max_tel</i>	maximum number of telescopes allowed
<i>ntel</i>	number of telescopes/detectors
<i>x</i>	X positions
<i>y</i>	Y positions
<i>z</i>	Z positions
<i>r</i>	radius of spheres including the whole devices

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.26 write_atmprof()

```
int write_atmprof (
    IO_BUFFER * iobuf,
    AtmProf * atmprof )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>atmprof</i>	Address of struct with relevant parts of atmospheric profile table

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.27 write_camera_layout()

```
int write_camera_layout (
    IO_BUFFER * iobuf,
    int itel,
    int type,
    int pixels,
    double * xp,
    double * yp )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>itel</i>	telescope number
<i>type</i>	camera type (hex/square)
<i>pixels</i>	number of pixels
<i>xp</i>	X positions of pixels
<i>yp</i>	Y position of pixels

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.28 write_input_lines()

```
int write_input_lines (
    IO_BUFFER * iobuf,
    struct linked_string * list )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>list</i>	starting point of linked list

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.29 write_photo_electrons()

```
int write_photo_electrons (
    IO_BUFFER * iobuf,
    int array,
    int tel,
    int npe,
    int flags,
    int pixels,
    int * pe_counts,
```



```

int * tstart,
double * t,
double * a,
int * photon_counts )

```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>array</i>	array number
<i>tel</i>	telescope number
<i>npe</i>	Total number of photo-electrons in the camera.
<i>pixels</i>	No. of pixels to be written
<i>flags</i>	Bit 0: save also amplitudes if available, Bit 1: p.e. list includes NSB p.e., bit 2: data also including no. of photons hitting each pixel. bit 3: photons (if any) are in wavelength range 300-550 nm.
<i>pe_counts</i>	Numbers of photo-electrons in each pixel
<i>tstart</i>	Offsets in 't' at which data for each pixel starts
<i>t</i>	Time of arrival of photons at the camera.
<i>a</i>	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
<i>photon_counts</i>	Optional number of photons arriving at a pixel (with flags bit 2 set)

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.30 write_shower_longitudinal()

```

int write_shower_longitudinal (
    IO_BUFFER * iobuf,
    int event,
    int type,
    double * data,
    int ndim,
    int np,
    int nthick,
    double thickstep )

```

See [telling_\(\)](#) in [iact.c](#) for more detailed parameter description.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>event</i>	event number
<i>type</i>	1 = particle numbers, 2 = energy, 3 = energy deposits
<i>data</i>	set of (usually 9) distributions
<i>ndim</i>	maximum number of entries per distribution
<i>np</i>	number of distributions (usually 9)
<i>nthick</i>	number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
<i>thickstep</i>	step size in g/cm**2

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.31 write_tel_array_end()

```
int write_tel_array_end (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int array )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (as opened in begin_write_tel_array())

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.32 write_tel_array_head()

```
int write_tel_array_head (
    IO_BUFFER * iobuf,
    IO_ITEM_HEADER * ih,
    int array )
```

Note: this function does not finish writing to the I/O block but after writing of the photons a call to [end_write_tel_array\(\)](#) is needed.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ih</i>	I/O item header (for item opened here)
<i>array</i>	Number of array

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.33 write_tel_block()

```
int write_tel_block (
    IO_BUFFER * iobuf,
    int type,
    int num,
    real * data,
    int len )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>type</i>	block type (see mc_tel.h)
<i>num</i>	Run or event number depending on type
<i>data</i>	Data as passed from CORSIKA
<i>len</i>	Number of elements to be written

Returns

0 (OK), -1, -2, -3 (error, as usual in eventio)

6.16.2.34 write_tel_compact_photons()

```
int write_tel_compact_photons (
    IO_BUFFER * iobuf,
    int array,
    int tel,
    double photons,
    struct compact_bunch * cbunches,
    int nbunches,
    int ext_bunches,
    char * ext_fname )
```

Usually, calls to this function for each telescope in an array should be enclosed within calls to [begin_write_tel_array\(\)](#) and [end_write_tel_array\(\)](#). This routine writes the more compact format (16 bytes per bunch). The more compact format should usually be used to save memory and disk space.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>array</i>	array number
<i>tel</i>	telescope number
<i>photons</i>	sum of photons (and fractions) in this device
<i>cbunches</i>	list of photon bunches
<i>nbunches</i>	number of elements in bunch list
<i>ext_bunches</i>	number of elements in external file
<i>ext_fname</i>	name of external (temporary) file

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.35 write_tel_offset()

```
int write_tel_offset (
    IO_BUFFER * iobuf,
```

```

    int narray,
    double toff,
    double * xoff,
    double * yoff )

```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.36 write_tel_offset_w()

```

int write_tel_offset_w (
    IO_BUFFER * iobuf,
    int narray,
    double toff,
    double * xoff,
    double * yoff,
    double * weight )

```

With respect to the backwards-compatible non-weights version [write_tel_offset\(\)](#), this version adds a weight to each offset position which should be normalized in such a way that with uniform sampling it should be the area over which showers are thrown divided by the number of array in each shower. With importance sampling the same relation should hold on average. So in either case, the average sum of weights for the different offsets in one shower equals just the area over which cores are randomized. This leaves the possibility to change the number of offsets from shower to shower.

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>narray</i>	Number of arrays of telescopes/detectors
<i>toff</i>	Time offset (ns, from first interaction to ground)
<i>xoff</i>	X offsets of arrays
<i>yoff</i>	Y offsets of arrays
<i>weight</i>	Area weight for uniform or importance sampled core offset.

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.37 write_tel_photons()

```
int write_tel_photons (
    IO_BUFFER * iobuf,
    int array,
    int tel,
    double photons,
    struct bunch * bunches,
    int nbunches,
    int ext_bunches,
    char * ext_fname )
```

Usually, calls to this function for each telescope in an array should be enclosed within calls to [begin_write_tel_array\(\)](#) and [end_write_tel_array\(\)](#). This routine writes the less compact format (32 bytes per bunch).

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>array</i>	array number
<i>tel</i>	telescope number
<i>photons</i>	sum of photons (and fractions) in this device
<i>bunches</i>	list of photon bunches
<i>nbunches</i>	number of elements in bunch list
<i>ext_bunches</i>	number of elements in external file
<i>ext_fname</i>	name of external (temporary) file

Returns

0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.16.2.38 write_tel_pos()

```
int write_tel_pos (
    IO_BUFFER * iobuf,
    int ntel,
    double * x,
    double * y,
    double * z,
    double * r )
```

Parameters

<i>iobuf</i>	I/O buffer descriptor
<i>ntel</i>	number of telescopes/detectors
<i>x</i>	X positions
<i>y</i>	Y positions
<i>z</i>	Z positions
<i>r</i>	radius of spheres including the whole devices

Returns

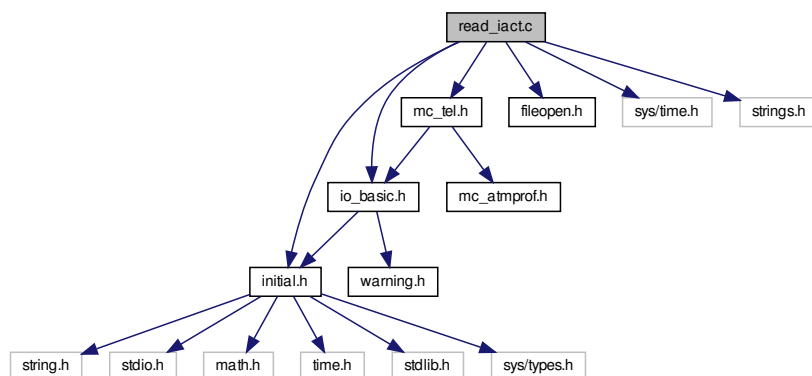
0 (o.k.), -1, -2, -3 (error, as usual in eventio)

6.17 read_iact.c File Reference

A program reading simulated CORSIKA data written through the IACT interface and shows the contents as readable text.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "fileopen.h"
#include <sys/time.h>
#include <strings.h>
```

Include dependency graph for read_iact.c:



Functions

- int [main](#) (int argc, char **argv)
Main program.
- int [my_print_hess_mc_phot](#) (IO_BUFFER *iobuf)
Print Monte Carlo photons and photo-electrons.
- void **syntax** (void)

6.17.1 Detailed Description

Relevant environment variables: PRINT_TEL_VERBOSE MAX_PRINT_ARRAY

Author

Konrad Bernloehr

Date

CVS \$Date: 2019/07/13 18:27:56 \$

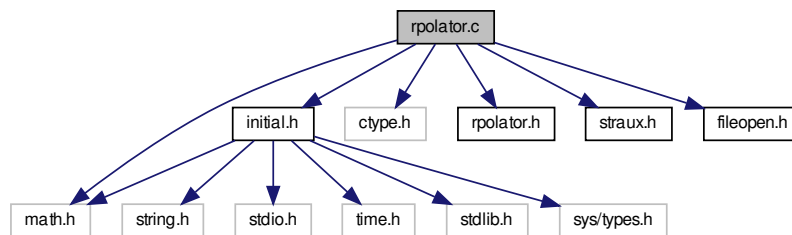
Version

CVS \$Revision: 1.2 \$

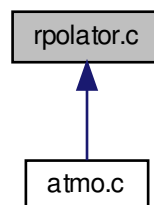
6.18 rpolator.c File Reference

Reading of configuration data tables and interpolation.

```
#include "initial.h"
#include <math.h>
#include <ctype.h>
#include "rpolator.h"
#include "straux.h"
#include "fileopen.h"
Include dependency graph for rpolator.c:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [rpt_list](#)

Registered interpolation tables allow re-use of tables without having to load them again.

Functions

- static double **csx** (double r, const [CsplinePar](#) *cp)
- static void [interp](#) (double x, double *v, int n, int *ipl, double *rpl)
Linear interpolation with binary search algorithm.
- struct [rpol_table](#) * [read_rpol1d_table](#) (const char *fn)

- Simplified version for loading a 1-d table with two columns.*

 - struct `rpol_table * read_rpol2d_table` (const char *fn, const char *ymarker)
- Simplified version for loading a 2-d table 1+ny columns (ny=available y values in header line).*

 - struct `rpol_table * read_rpol3d_table` (const char *fn)
- Simplified version for loading a 2-d table with x/y/z in three columns (x/y intervals rectangular).*

 - struct `rpol_table * read_rpol_table` (const char *fname, int nd, const char *ymarker, const char *options)
- General function for loading interpolation table combines 1-D and 2-D grid case.*

 - int `read_table` (const char *fname, int maxrow, double *col1, double *col2)
- Low-level reading of 2-column data tables up to given number of data rows.*

 - int `read_table2` (const char *fname, int maxrow, double *col1, double *col2)
 - int `read_table3` (const char *fname, int maxrow, double *col1, double *col2, double *col3)
- `read_table3()` and so on have more columns than `read_table` but are still only suitable for 1-D interpolation.*

 - int `read_table4` (const char *fname, int maxrow, double *col1, double *col2, double *col3, double *col4)
 - int `read_table5` (const char *fname, int maxrow, double *col1, double *col2, double *col3, double *col4, double *col5)
 - int `read_table_v` (const char *fname, FILE *fptr, size_t *nrow, size_t ncol, double ***col, const size_t *selcol)
- Read tables any length (up to some ridiculous maximum) with the requested columns either in natural order or picking columns as requested.*

 - double `rpol` (double *x, double *y, int n, double xp)
- Linear interpolation with binary search algorithm.*

 - double `rpol_2d_linear` (double *x, double *y, double *z, int nx, int ny, double xp, double yp, int eq, int clip)
- Linear interpolation in 2-D.*

 - double `rpol_2nd_order` (double *x, double *y, int n, double xp, int eq, int clip)
- Second/third order interpolation in 1-D with clipping option outside range.*

 - void `rpol_check_equi_range` (struct `rpol_table` *rpt)
 - double `rpol_cspline` (double *x, double *y, const `CsplinePar` *csp, int n, double xp, int eq, int clip)
- Cubic spline interpolation in 1-D with clipping option outside range.*

 - void `rpol_free` (struct `rpol_table` *rpt, int removing)
- Free a previously allocated interpolation table data structure.*

 - void `rpol_info` (struct `rpol_table` *rpt)
- Show information about given interpolation table.*

 - void `rpol_info_lvl` (struct `rpol_table` *rpt, int lvl)
- Report table info at given temporary verbosity level.*

 - int `rpol_is_verbose` (void)
- Report what verbosity level is set for the rpolator code.*

 - double `rpol_linear` (double *x, double *y, int n, double xp, int eq, int clip)
- Linear interpolation in 1-D with either direct access for equidistant table or with binary search algorithm.*

 - double `rpol_nearest` (double *x, double *y, int n, double xp, int eq, int clip)
- Nearest value (not actually interpolation) in 1-D with either direct access for equidistant table or with binary search algorithm.*

 - int `rpol_set_verbosity` (int lvl)
- Set the verbosity level for the rpolator code, return old level.*

 - double `rpolate` (struct `rpol_table` *rpt, double x, double y, int scheme)
- High-level interpolation function (user code only has to keep a pointer to the allocated object) generic for 1-D and 2-D tables.*

 - double `rpolate_1d` (struct `rpol_table` *rpt, double x, int scheme)
- High-level interpolation function (user code only has to keep a pointer to the allocated object) limited to 1-D table interpolation.*

 - double `rpolate_1d_lin` (struct `rpol_table` *rpt, double x)
- High-level interpolation function (user code only has to keep a pointer to the allocated object) limited to 1-D table interpolation, with linear interpolation scheme hard-wired (independent of any '#@RPOL@' header line).*

 - double `rpolate_2d` (struct `rpol_table` *rpt, double x, double y, int scheme)

High-level interpolation function (user code only has to keep a pointer to the allocated object) limited to 2-D table interpolation.

- `CsplinePar * set_1d_cubic_params` (double *x, double *y, int n, int clamped)
Set up cubic spline parameters for n-1 intervals resulting from n data points.
- `struct rpol_table * simple_rpol1d_table` (const char *label, double *x, double *y, int n, int clip)
A simplified way of setting up a 1-D rpol table for local use, without reading any files.
- `static void strip_comments` (char *line)

Variables

- `static int rpol_verbosity` = -1
- `static struct rpt_list * rpt_list_base`

6.18.1 Detailed Description

In contrast to the older low-level table reading and `rpol()` 1-D linear interpolation code, the function here allow flexible switching between 1-D and two different formats of 2-D tables, with support for different interpolation schemes (default being linear).

Author

Konrad Bernloehr

Date

2017

CVS \$Date: 2019/05/27 10:53:19 \$

Version

CVS \$Revision: 1.19 \$

6.18.2 Function Documentation

6.18.2.1 `interp()`

```
static void interp (
    double x,
    double * v,
    int n,
    int * ipl,
    double * rpl ) [static]
```

Linear interpolation between data point in sorted (i.e. monotonic ascending [no descending yet]) order. This function determines between which two data points the requested coordinate is and where between them. If the given coordinate is outside the covered range, the value for the corresponding edge is returned.

A binary search algorithm is used for fast interpolation.

Parameters

<i>x</i>	Input: the requested coordinate
<i>v</i>	Input: tabulated coordinates at data points
<i>n</i>	Input: number of data points
<i>ipl</i>	Output: the number of the data point following the requested coordinate in the given sorting ($1 \leq ipl \leq n-1$)
<i>rpl</i>	Output: the fraction $(x-v[ipl-1])/(v[ipl]-v[ipl-1])$ with $0 \leq rpl \leq 1$

References `rpol_table::x`.

6.18.2.2 read_rpol1d_table()

```
struct rpol_table * read_rpol1d_table (
    const char * fn )
```

The text free format input file is expected to contain (at least) two columns for x and y values. Any further columns are ignored. Comment and empty lines are ignored as well.

References `read_rpol_table()`.

6.18.2.3 read_rpol2d_table()

```
struct rpol_table * read_rpol2d_table (
    const char * fn,
    const char * ymarker )
```

The text input file is expected to contain a header line where, following the given marker text, the y values to which the following values correspond are listed. The number *ny* of distinct, ascending-order values in that line defines the number of *z* expected in the following data lines. The *x* value is the first value in each data line. If a data line contains more than $1+ny$ values the extra values are ignored. Comment (except header line) and empty lines are ignored.

References `read_rpol_table()`.

6.18.2.4 read_rpol3d_table()

```
struct rpol_table * read_rpol3d_table (
    const char * fn )
```

Functionally equivalent to `read_rpol2d_table()` but the y values are not given just once in a specially marked header line but repeated as the second value in each line. Instead of one header line plus *nx* data lines with $1+ny$ values the input to this function is expected to contain $nx*ny$ lines with three values each. While *x* and *y* values do not need to be equidistant, they are required to match a rectangular grid, with the same distinct, ascending-order *y* values appearing for each *x* value and the same distinct, ascending-order *x* values appearing for each *y* value, either as *x1/y1/z11*, *x2/y1/z21*, ... or *x1/y/z11*, *x1/y2/z12*, ...

References `read_rpol_table()`.

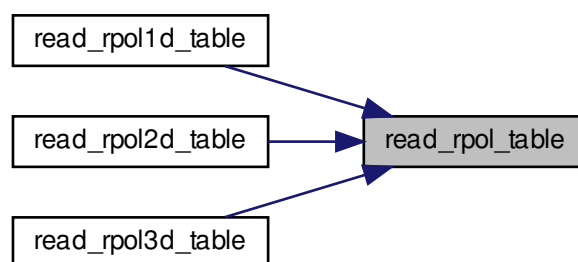
6.18.2.5 read_rpol_table()

```
struct rpol_table * read_rpol_table (
    const char * fname,
    int nd,
    const char * ymarker,
    const char * options )
```

Parameters

<i>fname</i>	Text input file name
<i>nd</i>	dimension/format parameter with the following possible values: 1: 1-D (2-column) input expected, 2: 2-D (1+ny columns) input with marker indicating special line for y values, 3: 2-D (3 columns) with repeated x and y values (matching rectangular grid), 0: format entirely defined in the data file, requiring the first line to start as '#@RPOL@', -1,-2,-3: If the first line starts as '#@RPOL@', this line defines the format and otherwise it is falling back to format 1, 2, or 3, respectively.
<i>ymarker</i>	The marker indicating the special header line listing the y values with nd=2 only (otherwise ignored).

Here is the caller graph for this function:



6.18.2.6 read_table()

```

int read_table (
    const char * fname,
    int maxrow,
    double * col1,
    double * col2 )

```

Parameters

<i>fname</i>	Name of file to be opened.
<i>maxrow</i>	Maximum number of (non-empty, non-comment) rows of data to read.
<i>col1</i>	Array where values of column 1 are to be copied to.
<i>col2</i>	Array where values of column 2 are to be copied to.

Returns

Number of data rows read (usable values in col1 and col2) or -1 (error).

6.18.2.7 read_table3()

```
int read_table3 (
    const char * fname,
    int maxrow,
    double * col1,
    double * col2,
    double * col3 )
```

6.18.2.8 read_table_v()

```
int read_table_v (
    const char * fname,
    FILE * fptr,
    size_t * nrow,
    size_t ncol,
    double *** col,
    const size_t * selcol )
```

All data areas are dynamically allocated (and must be fresh beforehand).

On memory allocation errors a -1 error code is returned but already allocated parts are not released and the file may still be opened.

Parameters

<i>fname</i>	Name or URL of file to read. @paran fptr File pointer if file already open or NULL if fname has to be opened.
<i>nrow</i>	Pointer to number of rows with valid data (pass address of a size_t variable). Input value used to guide initial allocation, not fixing actual rows to read.
<i>ncol</i>	Number of columns of data requested to be read.
<i>col</i>	Pointer to where pointers to column-wise data get allocated. (Need to pass address of a double ** variable.)
<i>selcol</i>	NULL for natural column order or pointer to ncol (natural, >=1) column numbers in selected order. Example: { 1, 7, 5 } will place data from the first column under (*col)[0], data from the seventh column under (*col)[1], and data from the fifth column under (*col)[2].

Returns

0 (OK), -1 (memory error), -2 (invalid parameter), -3 (asking for too many columns), -4 (too many rows of data, stopped reading).

6.18.2.9 rpol()

```
double rpol (
    double * x,
    double * y,
    int n,
    double xp )
```

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. This is the old-style function without any option for equidistant support points or clipping. Note that `rpol(px,py,n,xp)` is the same as `rpol_linear(px,py,n,xp,0,0)`.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value

Returns

Interpolated value

References [interp\(\)](#), `rpol_table::x`, and `rpol_table::y`.

6.18.2.10 rpol_2nd_order()

```
double rpol_2nd_order (
    double * x,
    double * y,
    int n,
    double xp,
    int eq,
    int clip )
```

Second to third order interpolation in 1-D with either direct access for equidistant table or with binary search algorithm. Instead of third order Lagrange interpolation it uses left- and right-sided 2nd order interpolation and a weighted mean between the two variants, rendering it effectively third order except for the intervals next to borders where it degenerates to 2nd order.

Higher-order interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval. Since there is no initialization phase, this is actually slower than the cubic spline interpolation.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points.
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Interpolated value

References `rpol_table::dxi`, `interp()`, `rpol_linear()`, `rpol_table::x`, and `rpol_table::y`.

6.18.2.11 rpol_cspline()

```
double rpol_cspline (
    double * x,
    double * y,
    const CsplinePar * csp,
    int n,
    double xp,
    int eq,
    int clip )
```

Cubic spline interpolation in 1-D with either direct access for equidistant table or with binary search algorithm.

Quadratic interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval. Because of the overhead of calculating the cubic spline parameters, those have to be initialized before the interpolation can be used. Initialisation has to be for either natural or clamped cubic splines.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

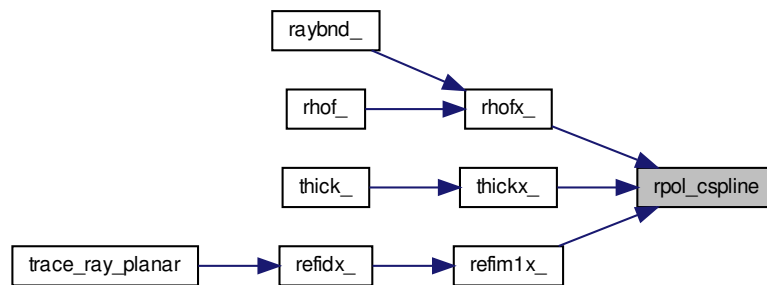
<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>csp</i>	Input: Cubic spline parameters (a,b,c,d) for each of n-1 intervals
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points.
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Interpolated value

References `interp()`, and `rpol_linear()`.

Here is the caller graph for this function:

**6.18.2.12 rpol_free()**

```
void rpol_free (
    struct rpol_table * rpt,
    int removing )
```

This is dangerous to use if the pointer is used in more than one place! Keep in mind that each time you ask to load the same table again you will just get a copy of the same pointer again. If that could be the case you better don't force deleting the structure itself.

References `rpol_table::csp`, `rpol_table::fname`, `rpol_table::options`, `rpol_table::use_count`, `rpol_table::x`, `rpol_table::y`, `rpol_table::z`, `rpol_table::zxmax`, and `rpol_table::zxmin`.

6.18.2.13 rpol_linear()

```
double rpol_linear (
    double * x,
    double * y,
    int n,
    double xp,
    int eq,
    int clip )
```

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval.

This function calls `interp()` to find out where to interpolate.

Parameters

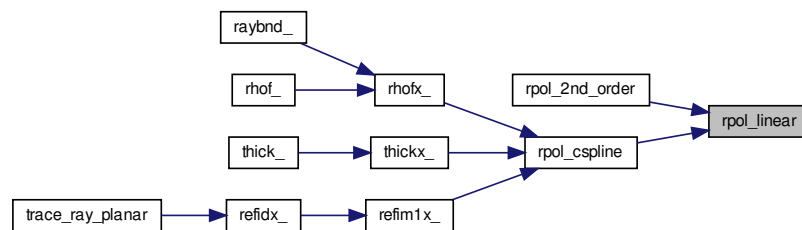
<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points.
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Interpolated value

References `rpol_table::dxi`, `interp()`, `rpol_table::x`, and `rpol_table::y`.

Here is the caller graph for this function:

**6.18.2.14 rpol_nearest()**

```
double rpol_nearest (
    double * x,
    double * y,
    int n,
    double xp,
    int eq,
    int clip )
```

Take the nearest data point in sorted (i.e. monotonic ascending [no descending yet]) order. The selected value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points. Generated by Doxygen
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Nearest value

References `rpol_table::dxi`, `interp()`, `rpol_table::x`, and `rpol_table::y`.

6.18.2.15 rpolate()

```
double rpolate (
    struct rpol_table * rpt,
    double x,
    double y,
    int scheme )
```

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it can represent either a 1-D or 2-D table.
<i>x</i>	The x coordinate value at which the 1-D or 2-D table is to be interpolated.
<i>y</i>	The y coordinate value at which a 2-D table is to be interpolated (ignored for 1-D). If non-zero for 1-D tables, a warning may be issued.
<i>scheme</i>	Interpolation scheme: 0 ... 4 (not all implemented) for a specific user-defined scheme or -1 (or other values outside of [0:4] range) for the scheme determined when the table was read and allocated.

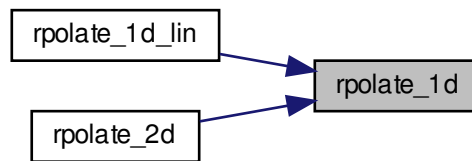
6.18.2.16 rpolate_1d()

```
double rpolate_1d (
    struct rpol_table * rpt,
    double x,
    int scheme )
```

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it should represent a 1-D table.
<i>x</i>	The x coordinate (abscissa) value at which the 1-D table is to be interpolated.
<i>scheme</i>	Interpolation scheme: 0 ... 4 for a specific user-defined scheme or -1 (or other values outside of [0:4] range) for the scheme determined when the table was read and allocated. For 2-D tables for which an upper envelope in x projection was requested a -1 scheme will interpolate in this upper envelope and -2 the lower envelope.

Here is the caller graph for this function:



6.18.2.17 rpolate_1d_lin()

```
double rpolate_1d_lin (
    struct rpol_table * rpt,
    double x )
```

This is the most direct equivalence to the older [rpol\(\)](#) function.

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it should represent a 1-D table.
<i>x</i>	The x coordinate (abscissa) value at which the 1-D table is to be interpolated.

References `rpolate_1d()`.

6.18.2.18 rpolate_2d()

```
double rpolate_2d (
    struct rpol_table * rpt,
    double x,
    double y,
    int scheme )
```

Fall-back to 1-D only after issuing a warning.

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it should represent a 2-D table.
<i>x</i>	The x coordinate value at which the 2-D table is to be interpolated.
<i>y</i>	The y coordinate value at which the 2-D table is to be interpolated (ignored for 1-D).
<i>scheme</i>	Interpolation scheme: 0 ... 4 (not all implemented) for a specific user-defined scheme or -1 (or other values outside of [0:4] range) for the scheme determined when the table was read and allocated.

References `rpol_table::fname`, `rpol_table::ndim`, and `rpolate_1d()`.

6.18.2.19 set_1d_cubic_params()

```
CsplinePar* set_1d_cubic_params (
    double * x,
    double * y,
    int n,
    int clamped )
```

The resulting cubic spline can either be a 'natural' one (second derivative is zero at the edges) or a clamped one (first derivative is fixed, currently to zero, at the edges).

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>clamped</i>	Input: 0 (natural cubic spline) or 1 (clamped cubic spline)

Returns

Allocated array of four parameters each defining the third order polynomial in each interval. In case of invalid input parameters a NULL pointer is returned.

References `cubic_params::d`, and `rpol_table::x`.

Here is the caller graph for this function:



6.18.2.20 simple_rpol1d_table()

```
struct rpol_table* simple_rpol1d_table (
    const char * label,
    double * x,
    double * y,
    int n,
    int clip )
```

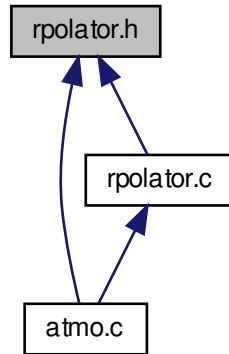
The returned pointer is also not hooked into the global linked list, thus is supposed to be safe to be removed after use.

References `rpol_table::clipping`, `rpol_table::fname`, `rpol_table::ndim`, `rpol_table::ny`, `rpol_table::scheme`, `rpol_table::use_count`, `rpol_table::x`, `rpol_table::y`, and `rpol_table::z`.

6.19 rpolator.h File Reference

Memory structure and interfaces for rpolator interpolation code.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [cubic_params](#)
Cubic spline interpolation (natural cubic splines = scheme 3, clamped cubic splines = scheme 4)
- struct [rpol_table](#)
Structure describing an interpolation table, interpolation scheme and selected options.

Macros

- `#define WITH_RPOLATOR 1`

Typedefs

- typedef struct [cubic_params](#) **CsplinePar**
- typedef struct [rpol_table](#) **RpolTable**

Functions

- struct [rpol_table](#) * [read_rpol1d_table](#) (const char *fn)
Simplified version for loading a 1-d table with two columns.
- struct [rpol_table](#) * [read_rpol2d_table](#) (const char *fn, const char *ymarker)
Simplified version for loading a 2-d table 1+ny columns (ny=available y values in header line).
- struct [rpol_table](#) * [read_rpol3d_table](#) (const char *fn)
Simplified version for loading a 2-d table with x/y/z in three columns (x/y intervals rectangular).
- struct [rpol_table](#) * [read_rpol_table](#) (const char *fname, int nd, const char *ymarker, const char *options)
General function for loading interpolation table combines 1-D and 2-D grid case.

- int [read_table](#) (const char *fname, int maxrow, double *col1, double *col2)
Low-level reading of 2-column data tables up to given number of data rows.
- int [read_table2](#) (const char *fname, int maxrow, double *col1, double *col2)
- int [read_table3](#) (const char *fname, int maxrow, double *col1, double *col2, double *col3)
[read_table3\(\)](#) and so on have more columns than [read_table](#) but are still only suitable for 1-D interpolation.
- int [read_table4](#) (const char *fname, int maxrow, double *col1, double *col2, double *col3, double *col4)
- int [read_table5](#) (const char *fname, int maxrow, double *col1, double *col2, double *col3, double *col4, double *col5)
- int [read_table_v](#) (const char *fname, FILE *fptr, size_t *nrow, size_t ncol, double ***col, const size_t *selcol)
Read tables any length (up to some ridiculous maximum) with the requested columns either in natural order or picking columns as requested.
- double [rpol](#) (double *x, double *y, int n, double xp)
Linear interpolation with binary search algorithm.
- double [rpol_2d_linear](#) (double *x, double *y, double *z, int nx, int ny, double xp, double yp, int eq, int clip)
Linear interpolation in 2-D.
- double [rpol_2nd_order](#) (double *x, double *y, int n, double xp, int eq, int clip)
Second/third order interpolation in 1-D with clipping option outside range.
- void [rpol_check_equi_range](#) (struct [rpol_table](#) *rpt)
- double [rpol_cspline](#) (double *x, double *y, const [CsplinePar](#) *csp, int n, double xp, int eq, int clip)
Cubic spline interpolation in 1-D with clipping option outside range.
- void [rpol_free](#) (struct [rpol_table](#) *rpt, int removing)
Free a previously allocated interpolation table data structure.
- void [rpol_info](#) (struct [rpol_table](#) *rpt)
Show information about given interpolation table.
- void [rpol_info_lvl](#) (struct [rpol_table](#) *rpt, int lvl)
Report table info at given temporary verbosity level.
- int [rpol_is_verbose](#) (void)
Report what verbosity level is set for the rpolator code.
- double [rpol_linear](#) (double *x, double *y, int n, double xp, int eq, int clip)
Linear interpolation in 1-D with either direct access for equidistant table or with binary search algorithm.
- double [rpol_nearest](#) (double *x, double *y, int n, double xp, int eq, int clip)
Nearest value (not actually interpolation) in 1-D with either direct access for equidistant table or with binary search algorithm.
- int [rpol_set_verbose](#) (int lvl)
Set the verbosity level for the rpolator code, return old level.
- double [rpolate](#) (struct [rpol_table](#) *rpt, double x, double y, int scheme)
High-level interpolation function (user code only has to keep a pointer to the allocated object) generic for 1-D and 2-D tables.
- double [rpolate_1d](#) (struct [rpol_table](#) *rpt, double x, int scheme)
High-level interpolation function (user code only has to keep a pointer to the allocated object) limited to 1-D table interpolation.
- double [rpolate_1d_lin](#) (struct [rpol_table](#) *rpt, double x)
High-level interpolation function (user code only has to keep a pointer to the allocated object) limited to 1-D table interpolation, with linear interpolation scheme hard-wired (independent of any '#@RPOL@' header line).
- double [rpolate_2d](#) (struct [rpol_table](#) *rpt, double x, double y, int scheme)
High-level interpolation function (user code only has to keep a pointer to the allocated object) limited to 2-D table interpolation.
- [CsplinePar](#) * [set_1d_cubic_params](#) (double *x, double *y, int n, int clamped)
Set up cubic spline parameters for n-1 intervals resulting from n data points.
- struct [rpol_table](#) * [simple_rpol1d_table](#) (const char *label, double *x, double *y, int n, int clip)
A simplified way of setting up a 1-D rpol table for local use, without reading any files.

6.19.1 Detailed Description

Author

Konrad Bernloehr

Date

2017

CVS \$Date: 2019/01/15 16:34:16 \$

Version

CVS \$Revision: 1.11 \$

6.19.2 Function Documentation

6.19.2.1 read_rpol1d_table()

```
struct rpol_table* read_rpol1d_table (
    const char * fn )
```

The text free format input file is expected to contain (at least) two columns for x and y values. Any further columns are ignored. Comment and empty lines are ignored as well.

References read_rpol_table().

6.19.2.2 read_rpol2d_table()

```
struct rpol_table* read_rpol2d_table (
    const char * fn,
    const char * ymarker )
```

The text input file is expected to contain a header line where, following the given marker text, the y values to which the following values correspond are listed. The number ny of distinct, ascending-order values in that line defines the number of z expected in the following data lines. The x value is the first value in each data line. If a data line contains more than 1+ny values the extra values are ignored. Comment (except header line) and empty lines are ignored.

References read_rpol_table().

6.19.2.3 read_rpol3d_table()

```
struct rpol_table* read_rpol3d_table (
    const char * fn )
```

Functionally equivalent to [read_rpol2d_table\(\)](#) but the y values are not given just once in a specially marked header line but repeated as the second value in each line. Instead of one header line plus nx data lines with 1+ny values the input to this function is expected to contain nx*ny lines with three values each. While x and y values do not need to be equidistant, they are required to match a rectangular grid, with the same distinct, ascending-order y values appearing for each x value and the same distinct, ascending-order x values appearing for each y value, either as x1/y1/z11, x2/y1/z21, ... or x1/y/z11, x1/y2/z12, ...

References [read_rpol_table\(\)](#).

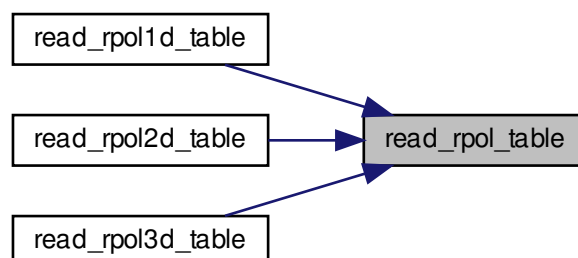
6.19.2.4 read_rpol_table()

```
struct rpol_table* read_rpol_table (
    const char * fname,
    int nd,
    const char * ymarker,
    const char * options )
```

Parameters

<i>fname</i>	Text input file name
<i>nd</i>	dimension/format parameter with the following possible values: 1: 1-D (2-column) input expected, 2: 2-D (1+ny columns) input with marker indicating special line for y values, 3: 2-D (3 columns) with repeated x and y values (matching rectangular grid), 0: format entirely defined in the data file, requiring the first line to start as '#@RPOL@', -1,-2,-3: If the first line starts as '#@RPOL@', this line defines the format and otherwise it is falling back to format 1, 2, or 3, respectively.
<i>ymarker</i>	The marker indicating the special header line listing the y values with nd=2 only (otherwise ignored).

Here is the caller graph for this function:



6.19.2.5 read_table()

```
int read_table (
    const char * fname,
    int maxrow,
    double * col1,
    double * col2 )
```

Parameters

<i>fname</i>	Name of file to be opened.
<i>maxrow</i>	Maximum number of (non-empty, non-comment) rows of data to read.
<i>col1</i>	Array where values of column 1 are to be copied to.
<i>col2</i>	Array where values of column 2 are to be copied to.

Returns

Number of data rows read (usable values in col1 and col2) or -1 (error).

6.19.2.6 read_table3()

```
int read_table3 (
    const char * fname,
    int maxrow,
    double * col1,
    double * col2,
    double * col3 )
```

6.19.2.7 read_table_v()

```
int read_table_v (
    const char * fname,
    FILE * fptr,
    size_t * nrow,
    size_t ncol,
    double *** col,
    const size_t * selcol )
```

All data areas are dynamically allocated (and must be fresh beforehand).

On memory allocation errors a -1 error code is returned but already allocated parts are not released and the file may still be opened.

Parameters

<i>fname</i>	Name or URL of file to read. @paran fptr File pointer if file already open or NULL if fname has to be opened.
<i>nrow</i>	Pointer to number of rows with valid data (pass address of a size_t variable). Input value used to guide initial allocation, not fixing actual rows to read.
<i>ncol</i>	Number of columns of data requested to be read.
<i>col</i>	Pointer to where pointers to column-wise data get allocated. (Need to pass address of a double ** variable.)
<i>selcol</i>	<p>NULL for natural column order or pointer to ncol (natural, >=1) column numbers in selected order. Example: { 1, 7, 5 } will place data from the first column under (*col)[0], data from the seventh column under (*col)[1], and data from the fifth column under (*col)[2].</p>

Returns

0 (OK), -1 (memory error), -2 (invalid parameter), -3 (asking for too many columns), -4 (too many rows of data, stopped reading).

6.19.2.8 rpol()

```
double rpol (
    double * x,
    double * y,
    int n,
    double xp )
```

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. This is the old-style function without any option for equidistant support points or clipping. Note that `rpoly(px,py,n,xp)` is the same as `rpoly_linear(px,py,n,xp,0,0)`.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value

Returns

Interpolated value

References [interp\(\)](#), `rpoly_table::x`, and `rpoly_table::y`.

6.19.2.9 rpol_2nd_order()

```
double rpol_2nd_order (
    double * x,
    double * y,
    int n,
    double xp,
    int eq,
    int clip )
```

Second to third order interpolation in 1-D with either direct access for equidistant table or with binary search algorithm. Instead of third order Lagrange interpolation it uses left- and right-sided 2nd order interpolation and a weighted mean between the two variants, rendering it effectively third order except for the intervals next to borders where it degenerates to 2nd order.

Higher-order interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval. Since there is no initialization phase, this is actually slower than the cubic spline interpolation.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points.
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Interpolated value

References `rpol_table::dxi`, `interp()`, `rpol_linear()`, `rpol_table::x`, and `rpol_table::y`.

6.19.2.10 rpol_cspline()

```
double rpol_cspline (
    double * x,
    double * y,
    const CsplinePar * csp,
    int n,
    double xp,
    int eq,
    int clip )
```

Cubic spline interpolation in 1-D with either direct access for equidistant table or with binary search algorithm.

Quadratic interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval. Because of the overhead of calculating the cubic spline parameters, those have to be initialized before the interpolation can be used. Initialisation has to be for either natural or clamped cubic splines.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

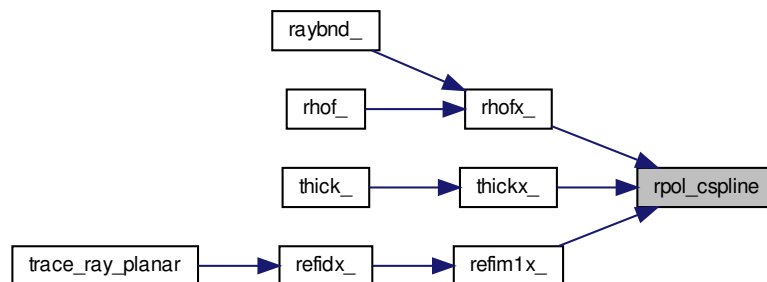
<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>csp</i>	Input: Cubic spline parameters (a,b,c,d) for each of n-1 intervals
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points.
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Interpolated value

References `interp()`, and `rpol_linear()`.

Here is the caller graph for this function:

**6.19.2.11 rpol_free()**

```
void rpol_free (
    struct rpol_table * rpt,
    int removing )
```

This is dangerous to use if the pointer is used in more than one place! Keep in mind that each time you ask to load the same table again you will just get a copy of the same pointer again. If that could be the case you better don't force deleting the structure itself.

References `rpol_table::csp`, `rpol_table::fname`, `rpol_table::options`, `rpol_table::use_count`, `rpol_table::x`, `rpol_table::y`, `rpol_table::z`, `rpol_table::zxmax`, and `rpol_table::zxmin`.

6.19.2.12 rpol_linear()

```
double rpol_linear (
    double * x,
    double * y,
    int n,
    double xp,
    int eq,
    int clip )
```

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval.

This function calls `interp()` to find out where to interpolate.

Parameters

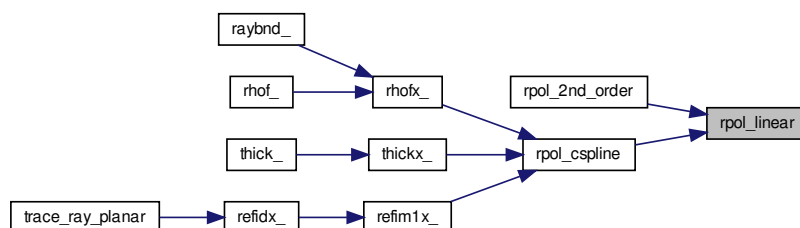
<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points.
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Interpolated value

References `rpol_table::dxi`, `interp()`, `rpol_table::x`, and `rpol_table::y`.

Here is the caller graph for this function:

**6.19.2.13 rpol_nearest()**

```
double rpol_nearest (
    double * x,
    double * y,
    int n,
    double xp,
    int eq,
    int clip )
```

Take the nearest data point in sorted (i.e. monotonic ascending [no descending yet]) order. The selected value is returned as a return value. If the table is known to be provided at equidistant supporting points, direct access is preferred. Otherwise a binary search algorithm is used to find the proper interval.

This function calls [interp\(\)](#) to find out where to interpolate.

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>xp</i>	Input: Coordinate of requested value
<i>eq</i>	Input: If non-zero: table is at equidistant points. Generated by Doxygen
<i>clip</i>	Input: Zero: no clipping; extrapolate with left/right edge value outside range. Non-zero: clip at edges; return 0. outside supported range.

Returns

Nearest value

References `rpol_table::dxi`, `interp()`, `rpol_table::x`, and `rpol_table::y`.

6.19.2.14 rpolate()

```
double rpolate (
    struct rpol_table * rpt,
    double x,
    double y,
    int scheme )
```

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it can represent either a 1-D or 2-D table.
<i>x</i>	The x coordinate value at which the 1-D or 2-D table is to be interpolated.
<i>y</i>	The y coordinate value at which a 2-D table is to be interpolated (ignored for 1-D). If non-zero for 1-D tables, a warning may be issued.
<i>scheme</i>	Interpolation scheme: 0 ... 4 (not all implemented) for a specific user-defined scheme or -1 (or other values outside of [0:4] range) for the scheme determined when the table was read and allocated.

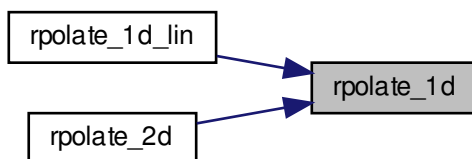
6.19.2.15 rpolate_1d()

```
double rpolate_1d (
    struct rpol_table * rpt,
    double x,
    int scheme )
```

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it should represent a 1-D table.
<i>x</i>	The x coordinate (abscissa) value at which the 1-D table is to be interpolated.
<i>scheme</i>	Interpolation scheme: 0 ... 4 for a specific user-defined scheme or -1 (or other values outside of [0:4] range) for the scheme determined when the table was read and allocated. For 2-D tables for which an upper envelope in x projection was requested a -1 scheme will interpolate in this upper envelope and -2 the lower envelope.

Here is the caller graph for this function:



6.19.2.16 rpolate_1d_lin()

```
double rpolate_1d_lin (
    struct rpol_table * rpt,
    double x )
```

This is the most direct equivalence to the older [rpol\(\)](#) function.

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it should represent a 1-D table.
<i>x</i>	The x coordinate (abscissa) value at which the 1-D table is to be interpolated.

References `rpolate_1d()`.

6.19.2.17 rpolate_2d()

```
double rpolate_2d (
    struct rpol_table * rpt,
    double x,
    double y,
    int scheme )
```

Fall-back to 1-D only after issuing a warning.

Parameters

<i>rpt</i>	Pointer to interpolation table structure, previously set up with <code>read_rpol_table</code> . Keep in mind that it gets only allocated once and, if you want to free it, you should not free it more than once. In the case of this function, it should represent a 2-D table.
<i>x</i>	The x coordinate value at which the 2-D table is to be interpolated.
<i>y</i>	The y coordinate value at which the 2-D table is to be interpolated (ignored for 1-D).
<i>scheme</i>	Interpolation scheme: 0 ... 4 (not all implemented) for a specific user-defined scheme or -1 (or other values outside of [0:4] range) for the scheme determined when the table was read and allocated.

References `rpol_table::fname`, `rpol_table::ndim`, and `rpolate_1d()`.

6.19.2.18 set_1d_cubic_params()

```
CsplinePar* set_1d_cubic_params (
    double * x,
    double * y,
    int n,
    int clamped )
```

The resulting cubic spline can either be a 'natural' one (second derivative is zero at the edges) or a clamped one (first derivative is fixed, currently to zero, at the edges).

Parameters

<i>x</i>	Input: Coordinates for data table
<i>y</i>	Input: Corresponding values for data table
<i>n</i>	Input: Number of data points
<i>clamped</i>	Input: 0 (natural cubic spline) or 1 (clamped cubic spline)

Returns

Allocated array of four parameters each defining the third order polynomial in each interval. In case of invalid input parameters a NULL pointer is returned.

References `cubic_params::d`, and `rpol_table::x`.

Here is the caller graph for this function:



6.19.2.19 simple_rpol1d_table()

```
struct rpol_table* simple_rpol1d_table (
    const char * label,
    double * x,
    double * y,
    int n,
    int clip )
```

The returned pointer is also not hooked into the global linked list, thus is supposed to be safe to be removed after use.

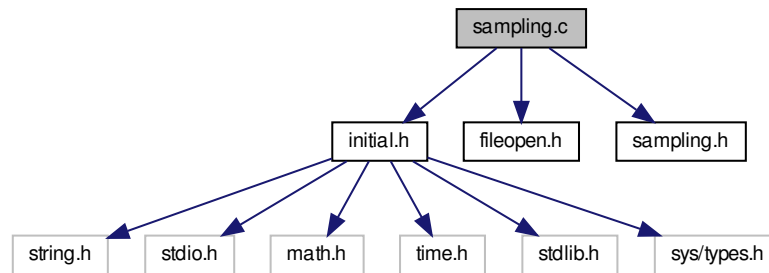
References `rpol_table::clipping`, `rpol_table::fname`, `rpol_table::ndim`, `rpol_table::ny`, `rpol_table::scheme`, `rpol_table::use_count`, `rpol_table::x`, `rpol_table::y`, and `rpol_table::z`.

6.20 sampling.c File Reference

Interface for importance sampled core offset distributions.

```
#include "initial.h"
#include "fileopen.h"
#include "sampling.h"
```

Include dependency graph for sampling.c:



Macros

- `#define rndm(i) iact_rndm(i)`

Functions

- double **iact_rndm** (int dummy)
- void **sample_offset** (const char ***sampling_fname**, double **core_range**, double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, double *xoff, double *yoff, double *sampling_area)

Get uniformly sampled or importance sampled offset of array with respect to core, in the plane perpendicular to the shower axis.

6.20.1 Detailed Description

Not used by default. The default distribution (per unit area) is flat.

Author

Konrad Bernloehr

Date

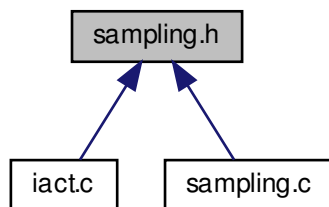
CVS \$Date: 2018/03/15 16:06:13 \$

Version

CVS \$Revision: 1.6 \$

6.21 `sampling.h` File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void `sample_offset` (const char *`sampling_fname`, double `core_range`, double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, double *xoff, double *yoff, double *sampling_↵_area)

Get uniformly sampled or importance sampled offset of array with respect to core, in the plane perpendicular to the shower axis.

6.22 `sim_skeleton.c` File Reference

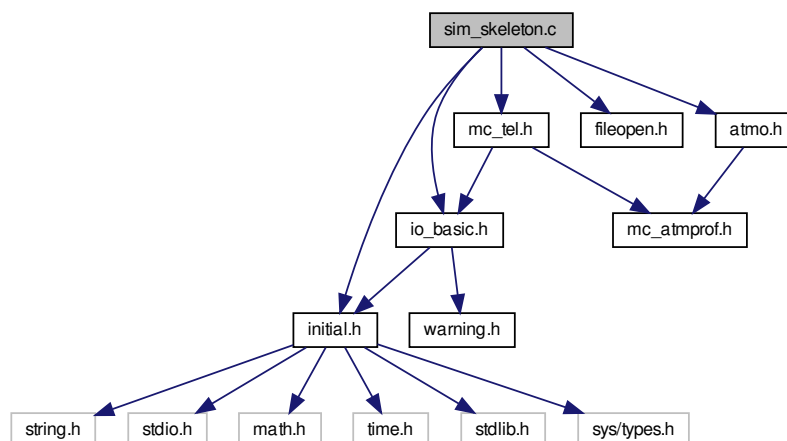
A (non-functional) skeleton program for reading CORSIKA IACT data.

```

#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "fileopen.h"
#include "atmo.h"

```

Include dependency graph for `sim_skeleton.c`:



Data Structures

- struct [camera_electronics](#)
Parameters of the electronics of a telescope.
- struct [mc_options](#)
Options of the simulation passed through to low-level functions.
- struct [mc_run](#)
Basic parameters of the CORSIKA run.
- struct [pm_camera](#)
Parameters of a telescope camera (pixels, ...)
- struct [simulated_shower_parameters](#)
Basic parameters of a simulated shower.
- struct [telescope_array](#)
Description of telescope position, array offsets and shower parameters.
- struct [telescope_optics](#)
Parameters describing the telescope optics.

Macros

- `#define MAX_ARRAY 100`
The largest no.
- `#define MAX_BUNCHES 2500000`
Change the following limits as appropriate <<<<
- `#define MAX_PHOTOELECTRONS 1000000` */** The largest number of photo-electrons. */*
- `#define MAX_PIXELS 1024`
The largest no.
- `#define MAX_TEL 16`
The largest no.
- `#define Nair(hkm) (1.+0.0002814*exp(-0.0947982*(hkm)-0.00134614*(hkm)*(hkm)))`
Refraction index of air as a function of height in km (0km<=h<=8km)

Functions

- double **atmospheric_transmission** (int iwl, double zem, double airmass)
- void **atmset_** (int *iatmo, double *obslev)
Set number of atmospheric model profile to be used.
- double **find_max_pos** (double *y, int n)
- double **heigh_** (double *x)
The CORSIKA built-in function for the height as a function of overburden.
- double **line_point_distance** (double x1, double y1, double z1, double cx, double cy, double cz, double x, double y, double z)
Distance between a straight line and a point in space.
- int **main** (int argc, char **argv)
Main program of Cherenkov telescope simulation.
- double **RandFlat** (void)
- double **rhof_** (double *h)
The CORSIKA built-in density lookup function.
- double **thick_** (double *h)
The CORSIKA built-in function for vertical atmospheric thickness (overburden).

Variables

- static double **airlightspeed** = 29.9792458/1.0002256
- struct [linked_string](#) **corsika_inputs**

6.22.1 Detailed Description

This file contains a (non-functional) skeleton of the telescope simulation. It serves only as an illustration of the essential usage of CORSIKA related eventio functions to read CORSIKA data in eventio format and how some of the required values are extracted. Comment lines with '...' usually indicate that you should fill in relevant code yourself.

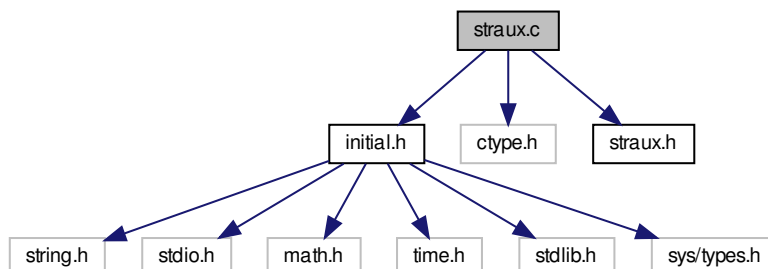
This file comes with no warranties.

If you want a working program using the same interfaces, look for `sim_telarray`.

6.23 straux.c File Reference

Check for abbreviations of strings and get words from strings.

```
#include "initial.h"
#include <ctype.h>
#include "straux.h"
Include dependency graph for straux.c:
```



Macros

- `#define NO_INITIAL_MACROS 1`

Functions

- int [abbrev](#) (CONST char *s, CONST char *t)
Compare strings s and t.
- int [getword](#) (CONST char *s, int *spos, char *word, int maxlen, char blank, char endchar)
*Copies a blank or '\0' or < endchar > delimited word from position *spos of the string s to the string word and increment *spos to the position of the first non-blank character after the word.*
- int [stricmp](#) (CONST char *a, CONST char *b)
Case independent comparison of character strings.

6.23.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2018/11/26 12:29:34 \$

Version

CVS \$Revision: 1.5 \$

6.23.2 Function Documentation

6.23.2.1 abbrev()

```
int abbrev (
    CONST char * s,
    CONST char * t )
```

s may be an abbreviation of t. Upper/lower case in s is ignored. s has to be at least as long as the leading upper case, digit, and '_' part of t.

Parameters

<i>s</i>	The string to be checked.
<i>t</i>	The test string with minimum part in upper case.

Returns

1 if s is an abbreviation of t, 0 if not.

6.23.2.2 getword()

```
int getword (
    CONST char * s,
    int * spos,
    char * word,
    int maxlen,
    char blank,
    char endchar )
```

The word must have a length less than or equal to maxlen.

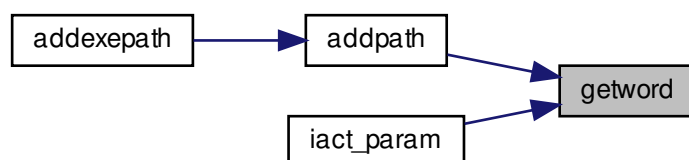
Parameters

<i>s</i>	string with any number of words.
<i>spos</i>	position in the string where we start and end.
<i>word</i>	the extracted word.
<i>maxlen</i>	the maximum allowed length of word.
<i>blank</i>	has the same effect as ' ', i.e. end-of-word.
<i>endchar</i>	his terminates the whole string (as '\0').

Returns

-2 : Invalid string or NULL -1 : The word was longer than maxlen (without the terminating '\0'); 0 : There were no more words in the string s. 1 : ok, we have a word and there are still more of them in the string s 2 : ok, but this was the last word

Here is the caller graph for this function:



6.23.2.3 stricmp()

```
int stricmp (
    CONST char * a,
    CONST char * b )
```

Parameters

<i>a,b</i>	– strings to be compared.
------------	---------------------------

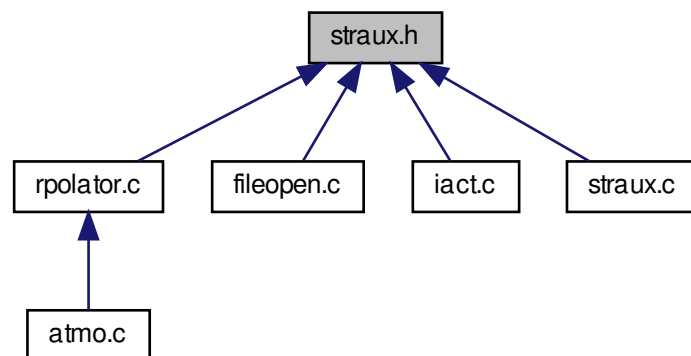
Returns

0 : strings are equal (except perhaps for case) >0 : a is lexically 'greater' than b <0 : a is lexically 'smaller' than b

6.24 straux.h File Reference

Check for abbreviations of strings and get words from strings.

This graph shows which files directly or indirectly include this file:



Macros

- `#define CONST const`

Functions

- `int abbrev (CONST char *s, CONST char *t)`
Compare strings s and t.
- `int getword (CONST char *s, int *spos, char *word, int maxlen, char blank, char endchar)`
*Copies a blank or '\0' or < endchar > delimited word from position *spos of the string s to the string word and increment *spos to the position of the first non-blank character after the word.*
- `int stricmp (CONST char *a, CONST char *b)`
Case independent comparison of character strings.

6.24.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2018/02/23 15:51:53 \$

Version

CVS \$Revision: 1.3 \$

6.24.2 Function Documentation

6.24.2.1 abbrev()

```
int abbrev (
    CONST char * s,
    CONST char * t )
```

s may be an abbreviation of t. Upper/lower case in s is ignored. s has to be at least as long as the leading upper case, digit, and '_' part of t.

Parameters

<i>s</i>	The string to be checked.
<i>t</i>	The test string with minimum part in upper case.

Returns

1 if s is an abbreviation of t, 0 if not.

6.24.2.2 getword()

```
int getword (
    CONST char * s,
    int * spos,
    char * word,
    int maxlen,
    char blank,
    char endchar )
```

The word must have a length less than or equal to maxlen.

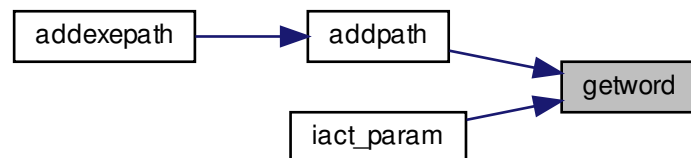
Parameters

<i>s</i>	string with any number of words.
<i>spos</i>	position in the string where we start and end.
<i>word</i>	the extracted word.
<i>maxlen</i>	the maximum allowed length of word.
<i>blank</i>	has the same effect as ' ', i.e. end-of-word.
<i>endchar</i>	his terminates the whole string (as '\0').

Returns

-2 : Invalid string or NULL -1 : The word was longer than maxlen (without the terminating '\0'); 0 : There were no more words in the string s. 1 : ok, we have a word and there are still more of them in the string s 2 : ok, but this was the last word

Here is the caller graph for this function:

**6.24.2.3 stricmp()**

```
int stricmp (
    CONST char * a,
    CONST char * b )
```

Parameters

<code>a,b</code>	– strings to be compared.
------------------	---------------------------

Returns

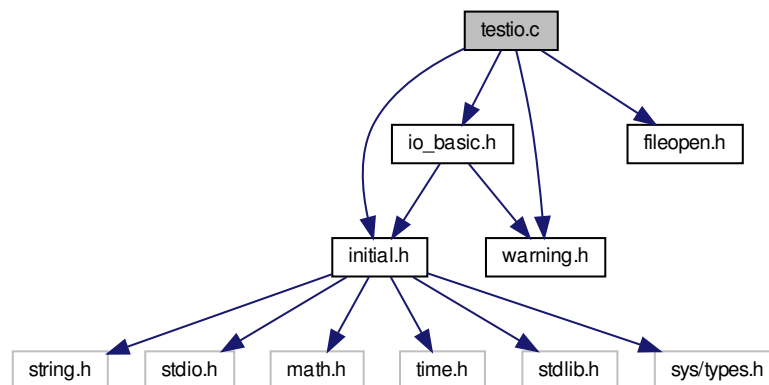
0 : strings are equal (except perhaps for case) >0 : a is lexically 'greater' than b <0 : a is lexically 'smaller' than b

6.25 testio.c File Reference

Test program for eventio data format.

```
#include "initial.h"
#include "warning.h"
#include "io_basic.h"
#include "fileopen.h"
```


Include dependency graph for testio.c:



Data Structures

- struct [test_struct](#)

Typedefs

- typedef struct [test_struct](#) **TEST_DATA**

Functions

- int [datacmp](#) ([TEST_DATA](#) *data1, [TEST_DATA](#) *data2)
Compare elements of test data structures.
- int [main](#) (int argc, char **argv)
Main function for I/O test program.
- int [read_test1](#) ([TEST_DATA](#) *data, [IO_BUFFER](#) *iobuf)
Read test data with single-element functions.
- int [read_test2](#) ([TEST_DATA](#) *data, [IO_BUFFER](#) *iobuf)
Read test data with vector functions as far as possible.
- int [read_test3](#) ([TEST_DATA](#) *data, [IO_BUFFER](#) *iobuf)
Read test data as a nested tree.
- int [read_test_lrg](#) ([IO_BUFFER](#) *iobuf)
Read large data block.
- void [syntax](#) (const char *prg)
Replacement for function missing on OS-9.
- int [write_test1](#) ([TEST_DATA](#) *data, [IO_BUFFER](#) *iobuf)
Write test data with single-element functions.
- int [write_test2](#) ([TEST_DATA](#) *data, [IO_BUFFER](#) *iobuf)
Write test data with vector functions as far as possible.
- int [write_test3](#) ([TEST_DATA](#) *data, [IO_BUFFER](#) *iobuf)
Write test data in nested items.
- int [write_test_lrg](#) ([IO_BUFFER](#) *iobuf)
Write large data block.

Variables

- static int **care_int**
- static int **care_long**
- static int **care_short**

6.25.1 Detailed Description

Author

Konrad Bernloehr

Date

1994 to 2010

CVS \$Date: 2019/07/12 15:37:40 \$

Version

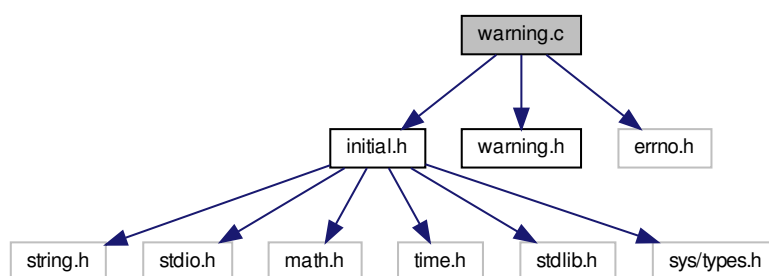
CVS \$Revision: 1.26 \$

6.26 warning.c File Reference

Pass warning messages to the screen or a usr function as set up.

```
#include "initial.h"
#include "warning.h"
#include <errno.h>
```

Include dependency graph for warning.c:



Data Structures

- struct [warn_specific_data](#)
A struct used to store thread-specific data.

Macros

- `#define __WARNING_MODULE 1`
- `#define get_warn_specific() (&warn_defaults)`

Functions

- void `flush_output` ()
Flush buffered output.
- void `set_aux_warning_function` (char *(*auxfunc)(void))
Set an auxilliary function for warnings.
- void `set_default_aux_warning_function` (char *(*auxfunc)(void))
- void `set_default_logging_function` (void(*user_function)(const char *, const char *, int, int))
- void `set_default_output_function` (void(*user_function)(const char *))
- int `set_default_warning` (int level, int mode)
- int `set_log_file` (const char *fname)
Set a new log file name and save it in local storage.
- void `set_logging_function` (void(*user_function)(const char *, const char *, int, int))
Set user-defined function for logging warnings and errors.
- void `set_output_function` (void(*user_function)(const char *))
Set a user-defined function as the function to be used for normal text output.
- int `set_warning` (int level, int mode)
Set a specific warning level and mode.
- void `warn_f_output_text` (const char *text)
Print a text string (without appending a newline etc.) on the screen or send it to a controlling process, depending on the setting of the output function.
- void `warn_f_warning` (const char *msgtext, const char *msgorigin, int msglevel, int msgno)
Issue a warning to screen or other configured target.
- void `warning_status` (int *plevel, int *pmode)
Inquire status of warning settings.

Variables

- static struct `warn_specific_data` `warn_defaults`

6.26.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2018/11/26 12:29:34 \$

Version

CVS \$Revision: 1.10 \$

One of the most import parameter for setting up the bevaviour is the warning level:

```
-----
Warning level: The lowest level of messages to be displayed
-----
Warning mode:
bit 0: display on screen (stderr),
bit 1: write to file,
bit 2: write with user-defined logging function.
bit 3: display origin if supplied.
bit 4: open log file for appending.
bit 5: call auxilliary function for time/date etc.
bit 6: use the auxilliary function output as origin string
      if no explicit origin was supplied.
bit 7: use syslog().
-----
```

6.26.2 Function Documentation

6.26.2.1 flush_output()

```
void flush_output (
    void )
```

Output is flushed, no matter if it is standard output or a special output function;

Returns

(none)

6.26.2.2 set_aux_warning_function()

```
void set_aux_warning_function (
    char *(*)(void) auxfunc )
```

This function may be used to insert time and date or origin etc. at the beginning of the warning text.

Parameters

<i>auxfunc</i>	– Pointer to a function taking no argument and returning a character string.
----------------	--

Returns

(none)

6.26.2.3 set_log_file()

```
int set_log_file (
    const char * fname )
```

If there was a log file with a different name opened previously, close it.

Parameters

<i>fname</i>	New name of log file for warnings
--------------	-----------------------------------

Returns

0 (o.k.), -1 (error)

6.26.2.4 set_logging_function()

```
void set_logging_function (
    void(*) (const char *, const char *, int, int) user_function )
```

Set a user-defined function as the function to be used for logging warnings and errors. To enable usage of this function, bit 2 of the warning mode must be set and other bits reset, if logging to screen and/or disk file is no longer wanted.

Parameter userfunc: Pointer to a function taking two strings (the message text and the origin text, which may be NULL) and two integers (message level and message number).

Returns

(none)

6.26.2.5 set_output_function()

```
void set_output_function (
    void(*) (const char *) user_function )
```

Such a function may be used to send output back to a remote control process via network.

Parameter userfunc: Pointer to a function taking a string (the text to be displayed) as argument.

Returns

(none)

6.26.2.6 set_warning()

```
int set_warning (
    int level,
    int mode )
```

Parameters

<i>level</i>	Warnings with level below this are ignored.
<i>mode</i>	To screen, to file, with user function ...

Returns

0 if ok, -1 if level and/or mode could not be set.

6.26.2.7 warn_f_output_text()

```
void warn_f_output_text (
    const char * text )
```

Parameters

<i>text</i>	A text string to be displayed.
-------------	--------------------------------

Returns

(none)

6.26.2.8 warn_f_warning()

```
void warn_f_warning (
    const char * msgtext,
    const char * msgorigin,
    int msglevel,
    int msgno )
```

Issue a warning to screen and/or file if the warning has a sufficiently large message 'level' (high enough severity). This function should best be called through the macros 'Information', 'Warning', and 'Error'. The name of this function has been changed from 'warning' to '_warning' to avoid trouble if you call 'warning' instead of 'Warning'. Now such a typo causes an error in the link step.

Parameters

<i>msgtext</i>	Warning or error text.
<i>msgorigin</i>	Optional origin (e.g. function name) or NULL.
<i>msglevel</i>	Level of message importance: negative: debugging if needed, 0-9: informative, 10-19: warning, 20-29: error.
<i>msgno</i>	Number of message or 0.

Returns

(none)

6.26.2.9 warning_status()

```
void warning_status (
    int * plevel,
    int * pmode )
```

Parameters

<i>plevel</i>	Pointer to variable for storing current level.
<i>pmode</i>	Pointer to store the current warning mode.

Returns

(none)

6.26.3 Variable Documentation

6.26.3.1 warn_defaults

```
struct warn_specific_data warn_defaults [static]
```

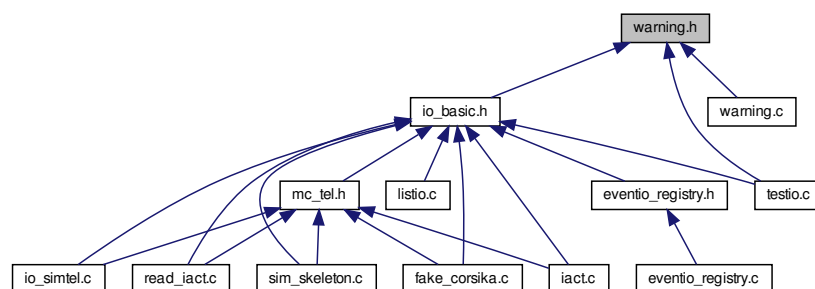
Initial value:

```
=
{
    0,
    1+8,
    "",
    "warning.log",
    "",
    0,
    NULL,
    NULL,
    NULL,
    NULL,
    0
}
```

6.27 warning.h File Reference

Pass warning messages to the screen or a usr function as set up.

This graph shows which files directly or indirectly include this file:



Macros

- `#define Error(string) warn_f_warning(string,WARNING_ORIGIN,20,0)`
- `#define Information(string) warn_f_warning(string,WARNING_ORIGIN,0,0)`
- `#define Output(string) warn_f_output_text(string)`
- `#define Warning(string) warn_f_warning(string,WARNING_ORIGIN,10,0)`
- `#define WARNING_ORIGIN (char *) NULL`

Functions

- void `flush_output` (void)
Flush buffered output.
- void `set_aux_warning_function` (char *(*auxfunc)(void))
Set an auxilliary function for warnings.
- void `set_default_aux_warning_function` (char *(*auxfunc)(void))
- void `set_default_logging_function` (void(*user_function)(const char *, const char *, int, int))
- void `set_default_output_function` (void(*user_function)(const char *))
- int `set_default_warning` (int level, int mode)
- int `set_log_file` (const char *fname)
Set a new log file name and save it in local storage.
- void `set_logging_function` (void(*user_function)(const char *, const char *, int, int))
Set user-defined function for logging warnings and errors.
- void `set_output_function` (void(*user_function)(const char *))
Set a user-defined function as the function to be used for normal text output.
- int `set_warning` (int level, int mode)
Set a specific warning level and mode.
- char * `warn_f_get_message_buffer` (void)
- void `warn_f_output_text` (const char *text)
Print a text string (without appending a newline etc.) on the screen or send it to a controlling process, depending on the setting of the output function.
- void `warn_f_warning` (const char *text, const char *origin, int level, int msgno)
Issue a warning to screen or other configured target.
- void `warning_status` (int *plevel, int *pmode)
Inquire status of warning settings.

6.27.1 Detailed Description

Author

Konrad Bernloehr

Date

CVS \$Date: 2010/07/20 13:37:45 \$

Version

CVS \$Revision: 1.5 \$

6.27.2 Function Documentation

6.27.2.1 flush_output()

```
void flush_output (
    void )
```

Output is flushed, no matter if it is standard output or a special output function;

Returns

(none)

6.27.2.2 set_aux_warning_function()

```
void set_aux_warning_function (
    char *(*)(void) auxfunc )
```

This function may be used to insert time and date or origin etc. at the beginning of the warning text.

Parameters

<i>auxfunc</i>	– Pointer to a function taking no argument and returning a character string.
----------------	--

Returns

(none)

6.27.2.3 set_log_file()

```
int set_log_file (
    const char * fname )
```

If there was a log file with a different name opened previously, close it.

Parameters

<i>fname</i>	New name of log file for warnings
--------------	-----------------------------------

Returns

0 (o.k.), -1 (error)

6.27.2.4 `set_logging_function()`

```
void set_logging_function (
    void(*) (const char *, const char *, int, int) user_function )
```

Set a user-defined function as the function to be used for logging warnings and errors. To enable usage of this function, bit 2 of the warning mode must be set and other bits reset, if logging to screen and/or disk file is no longer wanted.

Parameter *userfunc*: Pointer to a function taking two strings (the message text and the origin text, which may be NULL) and two integers (message level and message number).

Returns

(none)

6.27.2.5 `set_output_function()`

```
void set_output_function (
    void(*) (const char *) user_function )
```

Such a function may be used to send output back to a remote control process via network.

Parameter *userfunc*: Pointer to a function taking a string (the text to be displayed) as argument.

Returns

(none)

6.27.2.6 `set_warning()`

```
int set_warning (
    int level,
    int mode )
```

Parameters

<i>level</i>	Warnings with level below this are ignored.
<i>mode</i>	To screen, to file, with user function ...

Returns

0 if ok, -1 if level and/or mode could not be set.

6.27.2.7 `warn_f_output_text()`

```
void warn_f_output_text (
    const char * text )
```

Parameters

<i>text</i>	A text string to be displayed.
-------------	--------------------------------

Returns

(none)

6.27.2.8 warn_f_warning()

```
void warn_f_warning (
    const char * msgtext,
    const char * msgorigin,
    int msglevel,
    int msgno )
```

Issue a warning to screen and/or file if the warning has a sufficiently large message 'level' (high enough severity). This function should best be called through the macros 'Information', 'Warning', and 'Error'. The name of this function has been changed from 'warning' to '_warning' to avoid trouble if you call 'warning' instead of 'Warning'. Now such a typo causes an error in the link step.

Parameters

<i>msgtext</i>	Warning or error text.
<i>msgorigin</i>	Optional origin (e.g. function name) or NULL.
<i>msglevel</i>	Level of message importance: negative: debugging if needed, 0-9: informative, 10-19: warning, 20-29: error.
<i>msgno</i>	Number of message or 0.

Returns

(none)

6.27.2.9 warning_status()

```
void warning_status (
    int * plevel,
    int * pmode )
```

Parameters

<i>plevel</i>	Pointer to variable for storing current level.
<i>pmode</i>	Pointer to store the current warning mode.

Returns

(none)

Index

- `_STR_`
 - The CORSIKA iact/atmo interface, [13](#)
- `_struct_IO_BUFFER`, [49](#)
 - `buffer`, [50](#)
 - `buflen`, [50](#)
 - `byte_order`, [50](#)
 - `data`, [50](#)
 - `extended`, [50](#)
 - `input_file`, [50](#)
 - `input_fileno`, [51](#)
 - `is_allocated`, [51](#)
 - `item_extension`, [51](#)
 - `item_level`, [51](#)
 - `item_start_offset`, [51](#)
 - `output_file`, [51](#)
 - `output_fileno`, [51](#)
 - `sync_err_count`, [51](#)
 - `sync_err_max`, [52](#)
 - `user_function`, [52](#)
 - `w_remaining`, [52](#)
- `_struct_IO_ITEM_HEADER`, [52](#)
 - `can_search`, [53](#)
 - `ident`, [53](#)
 - `length`, [53](#)
 - `level`, [53](#)
 - `type`, [53](#)
 - `use_extension`, [53](#)
 - `user_flag`, [53](#)
 - `version`, [53](#)
- `abbrev`
 - `straux.c`, [216](#)
 - `straux.h`, [218](#)
- `addexepath`
 - `fileopen.c`, [95](#)
 - `fileopen.h`, [100](#)
- `addpath`
 - `fileopen.c`, [95](#)
 - `fileopen.h`, [101](#)
- `allocate_io_buffer`
 - `io_basic.h`, [119](#)
- `alt_km`
 - `atmospheric_profile`, [55](#)
- `altitude`
 - `telescope_array`, [76](#)
- `always_with_await`
 - `mc_options`, [61](#)
- `append_io_block_as_item`
 - `io_basic.h`, [119](#)
- `area`
 - `mc_run`, [63](#)
- `atime`
 - `photo_electron`, [66](#)
- `atm_init`
 - The CORSIKA iact/atmo interface, [14](#)
- `atmfit_`
 - The CORSIKA iact/atmo interface, [14](#)
- `atmnam_`
 - The CORSIKA iact/atmo interface, [15](#)
- `atmo.c`, [79](#)
- `atmo.h`, [84](#)
- `atmosphere`
 - `mc_run`, [63](#)
 - The CORSIKA iact/atmo interface, [32](#)
- `atmospheric_profile`, [54](#)
 - `alt_km`, [55](#)
 - `have_lay5_param`, [55](#)
 - `hlay`, [55](#)
 - `obslev`, [55](#)
- `atmprof_name`
 - The CORSIKA iact/atmo interface, [32](#)
- `atmset_`
 - The CORSIKA iact/atmo interface, [15](#)
 - The `sim_skeleton` program, [41](#)
- `aux`
 - `rpol_table`, [69](#)
- `await`
 - `simulated_shower_parameters`, [74](#)
 - `telescope_array`, [77](#)
- `azimuth`
 - `telescope_array`, [77](#)
- `begin_read_tel_array`
 - `io_simtel.c`, [143](#)
 - `mc_tel.h`, [169](#)
- `begin_write_tel_array`
 - `io_simtel.c`, [143](#)
 - `mc_tel.h`, [170](#)
- `bfield_bx`
 - `mc_run`, [64](#)
- `bfield_bz`
 - `mc_run`, [64](#)
- `bfield_rot`
 - `mc_run`, [64](#)
- `buffer`
 - `_struct_IO_BUFFER`, [50](#)
- `buflen`
 - `_struct_IO_BUFFER`, [50](#)
- `bunch`, [55](#)
- `bunchsize`
 - `mc_run`, [64](#)
- `byte_order`
 - `_struct_IO_BUFFER`, [50](#)
- `camera_electronics`, [56](#)
 - `simulated`, [56](#)
- `can_search`
 - `_struct_IO_ITEM_HEADER`, [53](#)
- `clear_shower_extra_parameters`
 - `io_simtel.c`, [144](#)
 - `mc_tel.h`, [170](#)

- clipping
 - rpol_table, 69
- clongi
 - simulated_shower_parameters, 74
- cmp_popen
 - fileopen.c, 96
- compact_bunch, 56
- compact_photon_hit
 - The CORSIKA iact/atmo interface, 15
- copy_item_to_io_block
 - io_basic.h, 120
- core_range
 - The CORSIKA iact/atmo interface, 32
- core_range1
 - The CORSIKA iact/atmo interface, 33
- corsika_version
 - mc_run, 64
 - The CORSIKA iact/atmo interface, 33
- csp
 - rpol_table, 69
- cubic_params, 57
- data
 - _struct_IO_BUFFER, 50
- datacmp
 - The testio program, 44
- dbl_to_sfloat
 - io_basic.h, 120
- detstruct, 57
- disable_permissive_pipes
 - fileopen.c, 96
 - fileopen.h, 101
- dmax
 - The CORSIKA iact/atmo interface, 33
- dxi
 - rpol_table, 69
- dyi
 - rpol_table, 69
- elongi
 - simulated_shower_parameters, 74
- enable_permissive_pipes
 - fileopen.c, 96
 - fileopen.h, 101
- end_read_tel_array
 - io_simtel.c, 144
 - mc_tel.h, 170
- end_write_tel_array
 - io_simtel.c, 144
 - mc_tel.h, 171
- equidistant
 - rpol_table, 69
- ev_reg_chain, 58
- ev_reg_entry, 59
- eventio_registered_typename
 - io_basic.h, 120
- eventio_registry.c, 86
 - find_ev_reg_std, 87
 - read_eventio_registry, 88
 - set_ev_reg_std, 88
- eventio_registry.h, 89
 - find_ev_reg_std, 90
 - read_eventio_registry, 90
 - set_ev_reg_std, 91
- exe_popen
 - fileopen.c, 97
- extend_io_buffer
 - io_basic.h, 120
- extended
 - _struct_IO_BUFFER, 50
- extprim_setup
 - The CORSIKA iact/atmo interface, 16
- extprm_
 - The CORSIKA iact/atmo interface, 16
- fake_corsika.c, 91
- fast_p_rho_rev
 - The CORSIKA iact/atmo interface, 33
- fileclose
 - fileopen.c, 97
 - fileopen.h, 102
- fileopen
 - fileopen.c, 97
 - fileopen.h, 102
- fileopen.c, 93
 - addexepath, 95
 - addpath, 95
 - cmp_popen, 96
 - disable_permissive_pipes, 96
 - enable_permissive_pipes, 96
 - exe_popen, 97
 - fileclose, 97
 - fileopen, 97
 - freeexepath, 98
 - freepath, 98
 - initpath, 98
 - listpath, 98
 - permissive_pipes, 99
 - root_exe_path, 99
 - root_path, 99
 - set_permissive_pipes, 98
 - uri_popen, 99
- fileopen.h, 99
 - addexepath, 100
 - addpath, 101
 - disable_permissive_pipes, 101
 - enable_permissive_pipes, 101
 - fileclose, 102
 - fileopen, 102
 - initpath, 102
 - listpath, 103
 - set_permissive_pipes, 103
- find_ev_reg_std
 - eventio_registry.c, 87
 - eventio_registry.h, 90
- find_io_block
 - io_basic.h, 121
- fltpt_to_sfloat

- io_basic.h, 121
- flush_output
 - warning.c, 224
 - warning.h, 229
- fn_rhof
 - The CORSIKA iact/atmo interface, 17
- fn_thick
 - The CORSIKA iact/atmo interface, 17
- fname
 - rpol_table, 69
- fparam
 - shower_extra_parameters, 72
- free_io_buffer
 - io_basic.h, 122
- freexepath
 - fileopen.c, 98
- freepath
 - fileopen.c, 98
- get_count
 - io_basic.h, 122
- get_count16
 - io_basic.h, 122
- get_count32
 - io_basic.h, 122
- get_double
 - io_basic.h, 123
- get_impact_offset
 - The CORSIKA iact/atmo interface, 17
- get_int32
 - io_basic.h, 123
- get_item_begin
 - io_basic.h, 123
- get_item_end
 - io_basic.h, 124
- get_long
 - io_basic.h, 124
- get_long_string
 - io_basic.h, 124
- get_real
 - io_basic.h, 124
- get_scount
 - io_basic.h, 125
- get_short
 - io_basic.h, 125
- get_string
 - io_basic.h, 125
- get_uint16
 - io_basic.h, 125
- get_uint32
 - io_basic.h, 126
- get_var_string
 - io_basic.h, 126
- get_vector_of_byte
 - io_basic.h, 126
- get_vector_of_uint16
 - io_basic.h, 126
- get_vector_of_uint16_scount_differential
 - io_basic.h, 127
- get_vector_of_uint32_scount_differential
 - io_basic.h, 127
- getword
 - straux.c, 216
 - straux.h, 219
- gridstruct, 59
- h1int
 - simulated_shower_parameters, 74
- have_lay5_param
 - atmospheric_profile, 55
- have_longi
 - simulated_shower_parameters, 74
- heigh_
 - The CORSIKA iact/atmo interface, 18
 - The fake_corsika program, 37
 - The sim_skeleton program, 42
- heighx_
 - The CORSIKA iact/atmo interface, 18
- high_E_model
 - mc_run, 64
- hlay
 - atmospheric_profile, 55
- hmax
 - simulated_shower_parameters, 75
- iact.c, 103
- iact.h, 109
- iact_param
 - The CORSIKA iact/atmo interface, 18
- id
 - shower_extra_parameters, 72
- ident
 - _struct_IO_ITEM_HEADER, 53
- impact_correction
 - The CORSIKA iact/atmo interface, 33
- in_detector
 - The CORSIKA iact/atmo interface, 19
- incpath, 60
- init_atmosphere_from_text_file
 - The CORSIKA iact/atmo interface, 19
- init_corsika_atmosphere
 - The CORSIKA iact/atmo interface, 19
- init_refraction_tables
 - The CORSIKA iact/atmo interface, 19
- init_shower_extra_parameters
 - io_simtel.c, 145
 - mc_tel.h, 171
- initial.h, 111
- initpath
 - fileopen.c, 98
 - fileopen.h, 102
- input_file
 - _struct_IO_BUFFER, 50
- input_fileno
 - _struct_IO_BUFFER, 51
- input_fname
 - mc_options, 62
- interp

- rpolator.c, 189
- io_basic.h, 113
 - allocate_io_buffer, 119
 - append_io_block_as_item, 119
 - copy_item_to_io_block, 120
 - dbl_to_sfloat, 120
 - eventio_registered_typename, 120
 - extend_io_buffer, 120
 - find_io_block, 121
 - fltp_to_sfloat, 121
 - free_io_buffer, 122
 - get_count, 122
 - get_count16, 122
 - get_count32, 122
 - get_double, 123
 - get_int32, 123
 - get_item_begin, 123
 - get_item_end, 124
 - get_long, 124
 - get_long_string, 124
 - get_real, 124
 - get_scount, 125
 - get_short, 125
 - get_string, 125
 - get_uint16, 125
 - get_uint32, 126
 - get_var_string, 126
 - get_vector_of_byte, 126
 - get_vector_of_uint16, 126
 - get_vector_of_uint16_scount_differential, 127
 - get_vector_of_uint32_scount_differential, 127
 - list_io_blocks, 127
 - list_sub_items, 127
 - next_subitem_ident, 128
 - next_subitem_length, 128
 - next_subitem_type, 128
 - put_byte, 119
 - put_count, 129
 - put_count16, 129
 - put_count32, 129
 - put_double, 130
 - put_int32, 130
 - put_item_begin, 130
 - put_item_begin_with_flags, 131
 - put_item_end, 131
 - put_long, 131
 - put_long_string, 132
 - put_real, 132
 - put_scount, 132
 - put_scount16, 133
 - put_scount32, 133
 - put_short, 133
 - put_string, 134
 - put_uint32, 134
 - put_var_string, 134
 - put_vector_of_byte, 135
 - put_vector_of_double, 135
 - put_vector_of_int, 135
 - put_vector_of_short, 135
 - put_vector_of_uint16, 135
 - read_io_block, 136
 - remove_item, 136
 - reset_io_block, 137
 - rewind_item, 137
 - search_sub_item, 137
 - set_eventio_registry_hook, 138
 - skip_io_block, 138
 - skip_subitem, 138
 - unget_item, 139
 - unput_item, 139
 - write_io_block, 139
- io_simtel.c, 140
 - begin_read_tel_array, 143
 - begin_write_tel_array, 143
 - clear_shower_extra_parameters, 144
 - end_read_tel_array, 144
 - end_write_tel_array, 144
 - init_shower_extra_parameters, 145
 - max_print, 160
 - print_atmprof, 145
 - print_camera_layout, 145
 - print_photo_electrons, 146
 - print_shower_longitudinal, 146
 - print_tel_block, 146
 - print_tel_offset, 146
 - print_tel_pos, 147
 - private_shower_extra_parameters, 161
 - read_atmprof, 147
 - read_camera_layout, 147
 - read_input_lines, 148
 - read_photo_electrons, 148
 - read_shower_longitudinal, 149
 - read_tel_array_end, 150
 - read_tel_array_head, 150
 - read_tel_block, 150
 - read_tel_offset, 151
 - read_tel_offset_w, 151
 - read_tel_photons, 152
 - read_tel_pos, 152
 - write_atmprof, 153
 - write_camera_layout, 153
 - write_input_lines, 154
 - write_photo_electrons, 154
 - write_shower_longitudinal, 155
 - write_tel_array_end, 155
 - write_tel_array_head, 157
 - write_tel_block, 157
 - write_tel_compact_photons, 158
 - write_tel_offset, 158
 - write_tel_offset_w, 159
 - write_tel_photons, 159
 - write_tel_pos, 160
- iobuf_max
 - mc_options, 62
- iparam
 - shower_extra_parameters, 72

- is_allocated
 - _struct_IO_BUFFER, 51
- is_set
 - shower_extra_parameters, 72
- item_extension
 - _struct_IO_BUFFER, 51
- item_level
 - _struct_IO_BUFFER, 51
- item_start_offset
 - _struct_IO_BUFFER, 51
- lambda
 - photo_electron, 66
- length
 - _struct_IO_ITEM_HEADER, 53
- level
 - _struct_IO_ITEM_HEADER, 53
- line_point_distance
 - The sim_skeleton program, 42
- linked_string, 61
- list_io_blocks
 - io_basic.h, 127
- list_sub_items
 - io_basic.h, 127
- listio.c, 161
- listpath
 - fileopen.c, 98
 - fileopen.h, 103
- logfname
 - warn_specific_data, 79
- low_E_model
 - mc_run, 64
- main
 - The listio program, 38
 - The read_iact program, 39
 - The testio program, 45
- MAX_ARRAY
 - The sim_skeleton program, 41
- MAX_BUNCHES
 - The sim_skeleton program, 41
- max_internal_bunches
 - The CORSIKA iact/atmo interface, 33
- max_io_buffer
 - The CORSIKA iact/atmo interface, 33
- MAX_PIXELS
 - The sim_skeleton program, 41
- max_print
 - io_simtel.c, 160
- MAX_TEL
 - The sim_skeleton program, 41
- mc_atmprof.c, 162
 - rho_fc, 163
- mc_atmprof.h, 164
 - rho_fc, 165
- mc_options, 61
 - always_with_await, 61
 - input_fname, 62
 - iobuf_max, 62
- mc_run, 62
 - area, 63
 - atmosphere, 63
 - bfield_bx, 64
 - bfield_bz, 64
 - bfield_rot, 64
 - bunchsize, 64
 - corsika_version, 64
 - high_E_model, 64
 - low_E_model, 64
 - num_arrays, 64
 - num_showers, 64
 - radius, 65
 - run, 65
 - start_depth, 65
 - viewcone_max, 65
 - viewcone_min, 65
- mc_tel.h, 166
 - begin_read_tel_array, 169
 - begin_write_tel_array, 170
 - clear_shower_extra_parameters, 170
 - end_read_tel_array, 170
 - end_write_tel_array, 171
 - init_shower_extra_parameters, 171
 - print_atmprof, 171
 - print_camera_layout, 172
 - print_photo_electrons, 172
 - print_shower_longitudinal, 172
 - print_tel_block, 172
 - print_tel_offset, 173
 - print_tel_pos, 173
 - read_atmprof, 173
 - read_camera_layout, 174
 - read_input_lines, 174
 - read_photo_electrons, 175
 - read_shower_longitudinal, 175
 - read_tel_array_end, 176
 - read_tel_array_head, 176
 - read_tel_block, 177
 - read_tel_offset, 177
 - read_tel_offset_w, 178
 - read_tel_photons, 178
 - read_tel_pos, 179
 - write_atmprof, 179
 - write_camera_layout, 179
 - write_input_lines, 180
 - write_photo_electrons, 180
 - write_shower_longitudinal, 181
 - write_tel_array_end, 182
 - write_tel_array_head, 182
 - write_tel_block, 182
 - write_tel_compact_photons, 183
 - write_tel_offset, 183
 - write_tel_offset_w, 184
 - write_tel_photons, 184
 - write_tel_pos, 185
- ndim
 - rpol_table, 69

- next_subitem_ident
 - io_basic.h, 128
- next_subitem_length
 - io_basic.h, 128
- next_subitem_type
 - io_basic.h, 128
- nfparam
 - shower_extra_parameters, 72
- Nint_f
 - The CORSIKA iact/atmo interface, 20
- niparam
 - shower_extra_parameters, 73
- num_arrays
 - mc_run, 64
- num_prof
 - The CORSIKA iact/atmo interface, 33
- num_showers
 - mc_run, 64
- ny
 - rpol_table, 69
- obslev
 - atmospheric_profile, 55
- options
 - rpol_table, 70
 - telescope_array, 77
- output_file
 - _struct_IO_BUFFER, 51
- output_fileno
 - _struct_IO_BUFFER, 51
- output_fname
 - The CORSIKA iact/atmo interface, 33
- p_alt
 - The CORSIKA iact/atmo interface, 34
- p_alt_rev
 - The CORSIKA iact/atmo interface, 34
- p_log_n1
 - The CORSIKA iact/atmo interface, 34
- permissive_pipes
 - fileopen.c, 99
- photo_electron, 65
 - atime, 66
 - lambda, 66
 - pixel, 66
- photon_bunch, 66
- photon_hit
 - The CORSIKA iact/atmo interface, 20
- pixel
 - photo_electron, 66
- pm_camera, 67
- print_atmprof
 - io_simtel.c, 145
 - mc_tel.h, 171
- print_camera_layout
 - io_simtel.c, 145
 - mc_tel.h, 172
- print_photo_electrons
 - io_simtel.c, 146
- mc_tel.h, 172
- print_shower_longitudinal
 - io_simtel.c, 146
 - mc_tel.h, 172
- print_tel_block
 - io_simtel.c, 146
 - mc_tel.h, 172
- print_tel_offset
 - io_simtel.c, 146
 - mc_tel.h, 173
- print_tel_pos
 - io_simtel.c, 147
 - mc_tel.h, 173
- private_shower_extra_parameters
 - io_simtel.c, 161
- put_byte
 - io_basic.h, 119
- put_count
 - io_basic.h, 129
- put_count16
 - io_basic.h, 129
- put_count32
 - io_basic.h, 129
- put_double
 - io_basic.h, 130
- put_int32
 - io_basic.h, 130
- put_item_begin
 - io_basic.h, 130
- put_item_begin_with_flags
 - io_basic.h, 131
- put_item_end
 - io_basic.h, 131
- put_long
 - io_basic.h, 131
- put_long_string
 - io_basic.h, 132
- put_real
 - io_basic.h, 132
- put_scount
 - io_basic.h, 132
- put_scount16
 - io_basic.h, 133
- put_scount32
 - io_basic.h, 133
- put_short
 - io_basic.h, 133
- put_string
 - io_basic.h, 134
- put_uint32
 - io_basic.h, 134
- put_var_string
 - io_basic.h, 134
- put_vector_of_byte
 - io_basic.h, 135
- put_vector_of_double
 - io_basic.h, 135
- put_vector_of_int

- io_basic.h, 135
- put_vector_of_short
 - io_basic.h, 135
- put_vector_of_uint16
 - io_basic.h, 135
- radius
 - mc_run, 65
- raybnd_
 - The CORSIKA iact/atmo interface, 20
- read_atmprof
 - io_simtel.c, 147
 - mc_tel.h, 173
- read_camera_layout
 - io_simtel.c, 147
 - mc_tel.h, 174
- read_eventio_registry
 - eventio_registry.c, 88
 - eventio_registry.h, 90
- read_iact.c, 186
- read_input_lines
 - io_simtel.c, 148
 - mc_tel.h, 174
- read_io_block
 - io_basic.h, 136
- read_photo_electrons
 - io_simtel.c, 148
 - mc_tel.h, 175
- read_rpol1d_table
 - rpolator.c, 190
 - rpolator.h, 202
- read_rpol2d_table
 - rpolator.c, 190
 - rpolator.h, 202
- read_rpol3d_table
 - rpolator.c, 190
 - rpolator.h, 202
- read_rpol_table
 - rpolator.c, 190
 - rpolator.h, 203
- read_shower_longitudinal
 - io_simtel.c, 149
 - mc_tel.h, 175
- read_table
 - rpolator.c, 191
 - rpolator.h, 203
- read_table3
 - rpolator.c, 191
 - rpolator.h, 204
- read_table_v
 - rpolator.c, 192
 - rpolator.h, 204
- read_tel_array_end
 - io_simtel.c, 150
 - mc_tel.h, 176
- read_tel_array_head
 - io_simtel.c, 150
 - mc_tel.h, 176
- read_tel_block
 - io_simtel.c, 150
 - mc_tel.h, 177
- read_tel_offset
 - io_simtel.c, 151
 - mc_tel.h, 177
- read_tel_offset_w
 - io_simtel.c, 151
 - mc_tel.h, 178
- read_tel_photons
 - io_simtel.c, 152
 - mc_tel.h, 178
- read_tel_pos
 - io_simtel.c, 152
 - mc_tel.h, 179
- read_test1
 - The testio program, 45
- read_test2
 - The testio program, 45
- read_test3
 - The testio program, 46
- read_test_lrg
 - The testio program, 46
- refidx_
 - The CORSIKA iact/atmo interface, 21
- refim1x_
 - The CORSIKA iact/atmo interface, 21
- refpos
 - telescope_array, 77
- remapped
 - rpol_table, 70
- remove_item
 - io_basic.h, 136
- reset_io_block
 - io_basic.h, 137
- rewind_item
 - io_basic.h, 137
- rhof_
 - The CORSIKA iact/atmo interface, 22
 - The fake_corsika program, 37
 - The sim_skeleton program, 43
- rhofc
 - mc_atmprof.c, 163
 - mc_atmprof.h, 165
- rhofx_
 - The CORSIKA iact/atmo interface, 22
- rmax
 - The CORSIKA iact/atmo interface, 34
- rndm
 - The CORSIKA iact/atmo interface, 23
- root_exe_path
 - fileopen.c, 99
- root_path
 - fileopen.c, 99
- rpol
 - rpolator.c, 192
 - rpolator.h, 205
- rpol_2nd_order
 - rpolator.c, 193

- rpolator.h, 205
- rpol_cspline
 - rpolator.c, 194
 - rpolator.h, 206
- rpol_free
 - rpolator.c, 195
 - rpolator.h, 207
- rpol_linear
 - rpolator.c, 195
 - rpolator.h, 207
- rpol_nearest
 - rpolator.c, 196
 - rpolator.h, 208
- rpol_table, 67
 - aux, 69
 - clipping, 69
 - csp, 69
 - dxi, 69
 - dyi, 69
 - equidistant, 69
 - fname, 69
 - ndim, 69
 - ny, 69
 - options, 70
 - remapped, 70
 - scheme, 70
 - use_count, 70
 - zxmax, 70
 - zxmin, 70
 - zxreq, 70
- rpolate
 - rpolator.c, 197
 - rpolator.h, 209
- rpolate_1d
 - rpolator.c, 197
 - rpolator.h, 209
- rpolate_1d_lin
 - rpolator.c, 198
 - rpolator.h, 210
- rpolate_2d
 - rpolator.c, 198
 - rpolator.h, 210
- rpolator.c, 187
 - interp, 189
 - read_rpol1d_table, 190
 - read_rpol2d_table, 190
 - read_rpol3d_table, 190
 - read_rpol_table, 190
 - read_table, 191
 - read_table3, 191
 - read_table_v, 192
 - rpol, 192
 - rpol_2nd_order, 193
 - rpol_cspline, 194
 - rpol_free, 195
 - rpol_linear, 195
 - rpol_nearest, 196
 - rpolate, 197
 - rpolate_1d, 197
 - rpolate_1d_lin, 198
 - rpolate_2d, 198
 - set_1d_cubic_params, 199
 - simple_rpol1d_table, 199
- rpolator.h, 200
 - read_rpol1d_table, 202
 - read_rpol2d_table, 202
 - read_rpol3d_table, 202
 - read_rpol_table, 203
 - read_table, 203
 - read_table3, 204
 - read_table_v, 204
 - rpol, 205
 - rpol_2nd_order, 205
 - rpol_cspline, 206
 - rpol_free, 207
 - rpol_linear, 207
 - rpol_nearest, 208
 - rpolate, 209
 - rpolate_1d, 209
 - rpolate_1d_lin, 210
 - rpolate_2d, 210
 - set_1d_cubic_params, 211
 - simple_rpol1d_table, 211
- rpt_list, 71
- run
 - mc_run, 65
- sample_offset
 - The CORSIKA iact/atmo interface, 23
- sampling.c, 212
- sampling.h, 213
- sampling_fname
 - The CORSIKA iact/atmo interface, 34
- scheme
 - rpol_table, 70
- search_sub_item
 - io_basic.h, 137
- set_1d_cubic_params
 - rpolator.c, 199
 - rpolator.h, 211
- set_aux_warning_function
 - warning.c, 224
 - warning.h, 229
- set_ev_reg_std
 - eventio_registry.c, 88
 - eventio_registry.h, 91
- set_eventio_registry_hook
 - io_basic.h, 138
- set_log_file
 - warning.c, 224
 - warning.h, 229
- set_logging_function
 - warning.c, 225
 - warning.h, 229
- set_output_function
 - warning.c, 225
 - warning.h, 230

- set_permissive_pipes
 - fileopen.c, [98](#)
 - fileopen.h, [103](#)
- set_random_systems
 - The CORSIKA iact/atmo interface, [24](#)
- set_system_displacement
 - The CORSIKA iact/atmo interface, [24](#)
- set_warning
 - warning.c, [225](#)
 - warning.h, [230](#)
- SHOW_MACRO
 - The CORSIKA iact/atmo interface, [13](#)
- shower_extra_parameters, [71](#)
 - fparam, [72](#)
 - id, [72](#)
 - iparam, [72](#)
 - is_set, [72](#)
 - nfparam, [72](#)
 - niparam, [73](#)
 - weight, [73](#)
- sim_skeleton.c, [213](#)
- simple_rpol1d_table
 - rpolator.c, [199](#)
 - rpolator.h, [211](#)
- simulated
 - camera_electronics, [56](#)
- simulated_shower_parameters, [73](#)
 - aweight, [74](#)
 - clongi, [74](#)
 - elongi, [74](#)
 - h1int, [74](#)
 - have_longi, [74](#)
 - hmax, [75](#)
 - step_longi, [75](#)
 - x0, [75](#)
 - xlongi, [75](#)
- skip_io_block
 - io_basic.h, [138](#)
- skip_subitem
 - io_basic.h, [138](#)
- source_altitude
 - telescope_array, [77](#)
- source_azimuth
 - telescope_array, [77](#)
- start_depth
 - mc_run, [65](#)
- step_longi
 - simulated_shower_parameters, [75](#)
- straux.c, [215](#)
 - abbrev, [216](#)
 - getword, [216](#)
 - stricmp, [217](#)
- straux.h, [217](#)
 - abbrev, [218](#)
 - getword, [219](#)
 - stricmp, [220](#)
- stricmp
 - straux.c, [217](#)
 - straux.h, [220](#)
- sync_err_count
 - _struct_IO_BUFFER, [51](#)
- sync_err_max
 - _struct_IO_BUFFER, [52](#)
- telasu_
 - The CORSIKA iact/atmo interface, [24](#)
- telend_
 - The CORSIKA iact/atmo interface, [25](#)
- telescope_array, [75](#)
 - altitude, [76](#)
 - aweight, [77](#)
 - azimuth, [77](#)
 - options, [77](#)
 - refpos, [77](#)
 - source_altitude, [77](#)
 - source_azimuth, [77](#)
 - toff, [77](#)
- telescope_optics, [78](#)
- televt_
 - The CORSIKA iact/atmo interface, [25](#)
- telfil_
 - The CORSIKA iact/atmo interface, [26](#)
- telinf_
 - The CORSIKA iact/atmo interface, [27](#)
- telling_
 - The CORSIKA iact/atmo interface, [27](#)
- tellni_
 - The CORSIKA iact/atmo interface, [28](#)
- telout_
 - The CORSIKA iact/atmo interface, [28](#)
- telpos_struct, [78](#)
- telrne_
 - The CORSIKA iact/atmo interface, [29](#)
- telrnh_
 - The CORSIKA iact/atmo interface, [30](#)
- telset_
 - The CORSIKA iact/atmo interface, [30](#)
- telshw_
 - The CORSIKA iact/atmo interface, [30](#)
- telsmp_
 - The CORSIKA iact/atmo interface, [30](#)
- test_struct, [78](#)
- testio.c, [220](#)
- The CORSIKA iact/atmo interface, [5](#)
 - _STR_, [13](#)
 - atm_init, [14](#)
 - atmfit_, [14](#)
 - atmnam_, [15](#)
 - atmosphere, [32](#)
 - atmprof_name, [32](#)
 - atmset_, [15](#)
 - compact_photon_hit, [15](#)
 - core_range, [32](#)
 - core_range1, [33](#)
 - corsika_version, [33](#)
 - dmax, [33](#)
 - extprim_setup, [16](#)

- extprm_, 16
- fast_p_rho_rev, 33
- fn_rhof, 17
- fn_thick, 17
- get_impact_offset, 17
- heigh_, 18
- heighx_, 18
- iact_param, 18
- impact_correction, 33
- in_detector, 19
- init_atmosphere_from_text_file, 19
- init_corsika_atmosphere, 19
- init_refraction_tables, 19
- max_internal_bunches, 33
- max_io_buffer, 33
- Nint_f, 20
- num_prof, 33
- output_fname, 33
- p_alt, 34
- p_alt_rev, 34
- p_log_n1, 34
- photon_hit, 20
- raybnd_, 20
- refidx_, 21
- refim1x_, 21
- rhof_, 22
- rhofx_, 22
- rmax, 34
- rndm, 23
- sample_offset, 23
- sampling_fname, 34
- set_random_systems, 24
- set_system_displacement, 24
- SHOW_MACRO, 13
- telasu_, 24
- telend_, 25
- televt_, 25
- telfil_, 26
- telinf_, 27
- telling_, 27
- tellni_, 28
- telout_, 28
- telrne_, 29
- telrnh_, 30
- telset_, 30
- telshw_, 30
- telsmp_, 30
- theta_central, 34
- thick_, 31
- thickx_, 31
- top_layer_2nd_altitude, 34
- top_layer_cfacs, 34
- top_layer_hscale, 34
- top_layer_hscale_rho0_cfacs_inv, 35
- top_layer_rho0, 35
- top_log_thickness, 35
- top_of_atm_table, 35
- trace_ray_planar, 32
- xtel, 35
- The fake_corsika program, 36
 - heigh_, 37
 - rhof_, 37
 - thick_, 37
- The listio program, 38
 - main, 38
- The read_iact program, 39
 - main, 39
- The sim_skeleton program, 40
 - atmset_, 41
 - heigh_, 42
 - line_point_distance, 42
 - MAX_ARRAY, 41
 - MAX_BUNCHES, 41
 - MAX_PIXELS, 41
 - MAX_TEL, 41
 - rhof_, 43
 - thick_, 43
- The testio program, 44
 - datacmp, 44
 - main, 45
 - read_test1, 45
 - read_test2, 45
 - read_test3, 46
 - read_test_lrg, 46
 - write_test1, 46
 - write_test2, 47
 - write_test3, 47
 - write_test_lrg, 47
- theta_central
 - The CORSIKA iact/atmo interface, 34
- thick_
 - The CORSIKA iact/atmo interface, 31
 - The fake_corsika program, 37
 - The sim_skeleton program, 43
- thickx_
 - The CORSIKA iact/atmo interface, 31
- toff
 - telescope_array, 77
- top_layer_2nd_altitude
 - The CORSIKA iact/atmo interface, 34
- top_layer_cfacs
 - The CORSIKA iact/atmo interface, 34
- top_layer_hscale
 - The CORSIKA iact/atmo interface, 34
- top_layer_hscale_rho0_cfacs_inv
 - The CORSIKA iact/atmo interface, 35
- top_layer_rho0
 - The CORSIKA iact/atmo interface, 35
- top_log_thickness
 - The CORSIKA iact/atmo interface, 35
- top_of_atm_table
 - The CORSIKA iact/atmo interface, 35
- trace_ray_planar
 - The CORSIKA iact/atmo interface, 32
- type
 - _struct_IO_ITEM_HEADER, 53

- unget_item
 - io_basic.h, 139
- unput_item
 - io_basic.h, 139
- uri_popen
 - fileopen.c, 99
- use_count
 - rpol_table, 70
- use_extension
 - _struct_IO_ITEM_HEADER, 53
- user_flag
 - _struct_IO_ITEM_HEADER, 53
- user_function
 - _struct_IO_BUFFER, 52
- version
 - _struct_IO_ITEM_HEADER, 53
- viewcone_max
 - mc_run, 65
- viewcone_min
 - mc_run, 65
- w_remaining
 - _struct_IO_BUFFER, 52
- warn_defaults
 - warning.c, 227
- warn_f_output_text
 - warning.c, 226
 - warning.h, 230
- warn_f_warning
 - warning.c, 226
 - warning.h, 231
- warn_specific_data, 79
 - logfname, 79
- warning.c, 222
 - flush_output, 224
 - set_aux_warning_function, 224
 - set_log_file, 224
 - set_logging_function, 225
 - set_output_function, 225
 - set_warning, 225
 - warn_defaults, 227
 - warn_f_output_text, 226
 - warn_f_warning, 226
 - warning_status, 226
- warning.h, 227
 - flush_output, 229
 - set_aux_warning_function, 229
 - set_log_file, 229
 - set_logging_function, 229
 - set_output_function, 230
 - set_warning, 230
 - warn_f_output_text, 230
 - warn_f_warning, 231
 - warning_status, 231
- warning_status
 - warning.c, 226
 - warning.h, 231
- weight
 - shower_extra_parameters, 73
- write_atmprof
 - io_simtel.c, 153
 - mc_tel.h, 179
- write_camera_layout
 - io_simtel.c, 153
 - mc_tel.h, 179
- write_input_lines
 - io_simtel.c, 154
 - mc_tel.h, 180
- write_io_block
 - io_basic.h, 139
- write_photo_electrons
 - io_simtel.c, 154
 - mc_tel.h, 180
- write_shower_longitudinal
 - io_simtel.c, 155
 - mc_tel.h, 181
- write_tel_array_end
 - io_simtel.c, 155
 - mc_tel.h, 182
- write_tel_array_head
 - io_simtel.c, 157
 - mc_tel.h, 182
- write_tel_block
 - io_simtel.c, 157
 - mc_tel.h, 182
- write_tel_compact_photons
 - io_simtel.c, 158
 - mc_tel.h, 183
- write_tel_offset
 - io_simtel.c, 158
 - mc_tel.h, 183
- write_tel_offset_w
 - io_simtel.c, 159
 - mc_tel.h, 184
- write_tel_photons
 - io_simtel.c, 159
 - mc_tel.h, 184
- write_tel_pos
 - io_simtel.c, 160
 - mc_tel.h, 185
- write_test1
 - The testio program, 46
- write_test2
 - The testio program, 47
- write_test3
 - The testio program, 47
- write_test_lrg
 - The testio program, 47
- x0
 - simulated_shower_parameters, 75
- xlongi
 - simulated_shower_parameters, 75
- xtel
 - The CORSIKA iact/atmo interface, 35
- zxmax

rpol_table, [70](#)
zxmin
 rpol_table, [70](#)
zxreq
 rpol_table, [70](#)