hessio

2023-05-31

Generated by Doxygen 1.9.1

Chapter 1

Introduction

1.1 Introduction to the eventio/hessio libraries.

The `hessio` libraries include a number of components which are heavily used in CORSIKA/sim_telarray (sim_↩
hessarray) simulations but also in some of the H.E.S.S. DAQ components. The basic components go back much further in history and were used for the DAQ of the CRT (Cosmic Ray Tracking) experiment, starting in 1991, and the HEGRA stereoscopic system of Cherenkov telescopes, starting in 1996. The library is thus also known under its orignal name: `eventio` library. The major components of the package include:

- The `eventio` data storage method with programming interfaces in C and C++.
- The `eventio` based high-level interfaces for shower simulations in the IACT interface to `CORSIKA`.
- The `eventio` based high-level interfaces for H.E.S.S. raw data and H.E.S.S./CTA simulations, as used by the `sim_telarray` program.
- A memory and speed efficient package for 1-D and 2-D histograms with full multi-threading support.
- The `eventio` based storage of the above histograms and conversion programs from the `eventio` format to PAW (HBOOK) and ROOT formats.
- A software run-time configuration interface named `hconfig` with a cpp-like preprocessor, also with full multi-threading support.

The `hessio` libraries are normally built in several variants:

- `libhessio` The variant optimised for single-threaded C programs. It has no multi-threading support and should not be used in multi-threaded DAQ environments. For simulations performed in a single thread, this variant provides optimum performance because no time is wasted in protecting critical sections by mutexes etc.
- `libhessio_r` The variant with full multi-threading support. Because of the overhead of protecting critical sections, it is not the optimal variant for single-threaded programs but (if linked with the POSIX threading library), will work for both multi-threaded and single-threaded programs. Linking: `-lhessio_↩
r -lpthread`
- `libhessio++` Like `libhessio` it offers no multi-threading support. In addition to `libhessio` it offers also the C++ interfaces to the `eventio` data format. As such, it requires linking with the C++ Standard Library. Single-threaded C++ programs would normally be linked against this variant: `-lhessio++`

- `libhessio++_r` offers everything of `libhessio_r` plus the C++ interfaces to the `eventio` data format. Multi-threaded C++ programs would normally be linked against this variant: `-lhessio++_↩ r -lpthread`

All of these libraries can be built as shared libraries and as static libraries, thus adding up to a total of eight libraries installed. Depending on definitions in the Makefile, the building of static libraries may be skipped by default.

The main documentation web page for this module can be found at
`https://www.mpi-hd.mpg.de/hfm/~bernlohr/HESS/Software/hessio/` (HESS-internal) or at
`https://www.mpi-hd.mpg.de/hfm/CTA/MC/Software/Doc/hessio/` (CTA-internal).

1.2 Eventio format documentation

The underlying `eventio` data format and the C and C++ programming interfaces are documented separately. See `http://www.mpi-hd.mpg.de/hfm/~bernlohr/HESS/Software/hessio/eventio_en.pdf` or `https://www.mpi-hd.mpg.de/hfm/CTA/MC/Software/Doc/hessio/eventio_en.pdf`.

1.3 Utility and test programs in the hessio module

A `make install` in the hessio module will, apart from the different variants of the library, install a number of programs. These include

- testio: A test program for the C programming interface. Should be run once if you go to a new platform or compiler.
- TestIO: A test program for the C++ programming interface. Should be run once if you go to a new platform or compiler. The output file generated should also be bitwise identical to that from the C interface test program.
- listio: Lists `eventio` data blocks in a data file or stream. Can also show the sub-block hierarchy.
- statio: Count the number of `eventio` top-level data blocks of each type and the total amount of (uncompressed) data for each block type. Also showing the version numbers involved.
- filterio: Select or deselect given types of `eventio` top-level data blocks between input or output, not requiring any support for the structure of the data block types.
- fcat: Like the standard 'cat' program but accepting any file type known by the fileopen() function as input, with decompression as implied by the filetype extension.
- read_hess: Reads output files generated by `sim_telarray` (aka `sim_hessarray`) and may optionally redo the image cleaning and shower reconstruction. It may be most useful to quickly visualize the images in the data file. Also called read_cta or read_simtel. In addition to `sim_telarray` it can also show the contents of CORSIKA/IACT files.
- read_hess_nr: A variant of read_hess.c without the reconstruction and analysis code, only suitable for showing the contents of sim_telarray output.
- read_iact: A minimal program to show the contents of CORSIKA/IACT files.
- gen_lookup: Process the histograms generated by read_hess to obtain lookup tables for width, length, energy, angular resolution, etc., which are used for further processing with read_hess.
- list_histograms: Show histograms embedded into an `eventio` file which can be either a dedicated histogram file or a general data file with any number of histogram blocks. It can also do projections of 2-D histograms or show the ratio of the contents of two histograms.

- add_histograms: Add up multiple occurences of matching histograms (in ID, type, limits, and size) from one or multiple files into a new histogram file, independent of any format conversion.
- hdata2hbook: Converts from the `eventio` histogram format to the now-outdated HBOOK/Paw format. Histogram blocks can be anywhere in a data file. You can also add up identical histograms from different input files before exporting.
- hdata2root: Converts from the `eventio` histogram format to the ROOT format. Like `hdata2hbook`.
- merge_simtel: Combine telescope data for corresponding events from two separate telescope simulations into a common file, for all telescopes or specified subsets.
- extract_simtel: Similar to `merge_simtel.c` for extracting data on subsets of telescopes, but just from one input file rather than two.
- extract_calibevent: Utility program for extracting the dark/pedestal/lid-LED/flatfield type calibration events produced internally in `sim_telarray` (which get wrapped into dedicated data blocks) into normal events for further analysis.
- split_hessio: Split up to single data stream produced by `sim_telarray` for a whole array of telescopes into separate files for individual telescope data and monitoring etc. as well as for MC true data and central trigger information, more closely corresponding to actual data streams recorded.
- gen_trgmask: Fixing a problem with 2012/13 versions of sim_telarray for camera configurations with multiple types of triggers where the information on which type of trigger fired got lost. This tool recovers this information from the log files. Not needed for new simulations (nor for old ones which could only have one type of trigger).
- check_trgmask: Check the camera trigger type bit patterns generated by the gen_trgmask tool for consistency.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 The add_histograms program

Functions

- void **syntax** (const char ∗prgm)

5.1.1 Detailed Description

5.2 The best_of program

One type is before the addition of 68% and 80% angular resolution values.

Data Structures

- struct best_value

Enumerations

- enum **SpecType** {
  SPEC_NONE = -1 , **SPEC_GAMMA** = 0 , **SPEC_ELECTRON** = 1 , **SPEC_PROTON** = 101 ,
  SPEC_HE = 402 , **SPEC_CNO** = 1407 , **SPEC_SI** = 2814 , **SPEC_IRON** = 5626 }
- enum **espec_t** { **OLD_E_POWERLAW** = 1 , **NEW_E_POWERLAW** = 2 , **NEW_E_PL_LGN1** = 3 , **NEW_**↩
  E_PL_LGN2 = 4 }
- enum **BestChoice** {
  BestDiff =1 , **BestIntegral** =2 , **BestAngle** =3 , **BestEres** =4 ,
  BestRate =5 , **BestCombined** =6 , **BestAll** =7 }

Functions

- string **particle_type** (SpecType sp)
- double **Crab_Unit** (double E)
- static double **cu** (double x)
- double **Crab_Unit_int** (double E)
- double **ergs** (double E)
- static double **f50** (double x)
- static double **fsp50** (double x)
- double **Flux_req50_south** (double E)
- double **Flux_req50_E2erg_south** (double E)
- double **Flux_req50_CU_south** (double E)
- static double **fn50** (double x)
- static double **fnsp50** (double x)
- double **Flux_req50_north** (double E)
- double **Flux_req50_E2erg_north** (double E)
- double **Flux_req50_CU_north** (double E)
- static double **f5** (double x)
- static double **fsp5** (double x)
- double **Flux_req5_south** (double E)
- double **Flux_req5_E2erg_south** (double E)
- double **Flux_req5_CU_south** (double E)
- static double **fn5** (double x)
- static double **fnsp5** (double x)
- double **Flux_req5_north** (double E)
- double **Flux_req5_E2erg_north** (double E)
- double **Flux_req5_CU_north** (double E)
- static double **f05** (double x)
- static double **fsp05** (double x)
- double **Flux_req05_south** (double E)
- double **Flux_req05_E2erg_south** (double E)
- double **Flux_req05_CU_south** (double E)
- static double **fn05** (double x)
- static double **fnsp05** (double x)
- double **Flux_req05_north** (double E)
- double **Flux_req05_E2erg_north** (double E)
- double **Flux_req05_CU_north** (double E)
- static double **fd50** (double x)
- static double **fdes50** (double x)
- double **Flux_goal50_south** (double E)
- double **Flux_goal50_E2erg_south** (double E)
- double **Flux_goal50_CU_south** (double E)
- static double **fnd50** (double x)
- static double **fndes50** (double x)
- double **Flux_goal50_north** (double E)
- double **Flux_goal50_E2erg_north** (double E)
- double **Flux_goal50_CU_north** (double E)
- double **Angular_resolution_req** (double E)
- double **Angular_resolution_goal** (double E)
- static double **eresb** (double E)
- double **Energy_resolution_req** (double E)
- static double **eresdb** (double E)
- double **Energy_resolution_goal** (double E)
- double **flux_int** (SpecType sp, double E1, double E2)
- double **lima17** (double on, double off, double alpha)

- bool **matching_required_diffsens** (int calc_pput, bool with_flux, double E, double diff_sens)
- bool **matching_required_performance** (int calc_pput, bool with_flux, double E, double diff_sens, double angres, double eres)
- bool **matching_required_angres** (double E, double angres)
- bool **matching_required_eres** (double E, double eres)
- int **main** (int argc, char ∗∗argv)

Variables

- static double **sce** = 1.6022
- static double **sca** = 1e-4
- static double **sc** = sce∗sca
- espec_t **espec_type** = OLD_E_POWERLAW

5.2.1 Detailed Description

One type is before the addition of 68% and 80% angular resolution values.

Another one is after addition of angular resolution but before addition of the energy resolution, and the third one is after the energy resolution got added to the output. The different formats are recognized by the presence and position of the histogram number (12056 to 12064 normally) on which the sensitivity evaluation is mainly based.

5.3 The fcat program

Macros

- #define **BSIZE** 8192

Functions

- void **syntax** (void)
- int **main** (int argc, char ∗∗argv)

5.3.1 Detailed Description

5.4 The list_histogram program

Functions

- long project_histogram (long ihisto, int proj)

  Project a 2-D histogram onto one of its axes.
- void print_ratio (HISTOGRAM ∗histo1, HISTOGRAM ∗histo2, double fact)

  Print ratio of two histograms: fact ∗ histo1 / histo2.
- int main (int argc, char ∗∗argv)

  Main program.

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 print_ratio()

```
void print_ratio (
             HISTOGRAM * histo1,
             HISTOGRAM * histo2,
             double fact )
```

Print ratio of two histograms: fact ∗ histo1 / histo2.

Parameters

histo1	Pointer to histogram 1
histo2	Pointer to histogram 2
fact	Scaling factor

Returns

(none)

References histogram::counts, histogram::entries, histogram::extension, histogram::ident, Histogram_Parameters←↩ ::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, Histogram_Parameters::real, histogram::tentries, histogram::title, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

5.4.2.2 project_histogram()

```
long project_histogram (
             long ihisto,
             int proj )
```

Project a 2-D histogram onto one of its axes.

Parameters

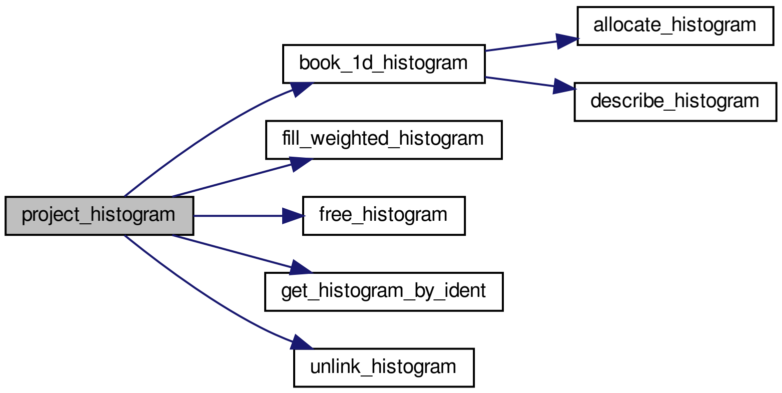
ihisto	ID of 2-D histogram to be projected
proj	1 for projection on axis 1, 2 for projection on axis 2.

Returns

Histogram ID of new registered histogram with projection results or 0.

References book_1d_histogram(), histogram::extension, fill_weighted_histogram(), free_histogram(), get_↩ histogram_by_ident(), Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram↩ _Parameters::real, histogram::title, histogram::type, unlink_histogram(), and Histogram_Parameters::upper_limit.

Here is the call graph for this function:



5.5 The iact_2d-to-3d program

Data Structures

- struct selector

Macros

- #define **MAXTEL** 5
- #define **MAXTEL** 5

Typedefs

- typedef struct selector **Selector**

Functions

- void **ioerrorcheck** (void)
- int tel_conv_mc_phot (EventIO &evio)

  Convert photon bunches from a single telescope.
- int array_conv_mc_phot (EventIO &evio)

  Convert photon bunches from a full array of telescopes.
- void **syntax** (const string &prg)
- int main (int argc, char ∗∗argv)

  Main program.

- int array_select_mc_phot (IO_BUFFER ∗iobuf)

  Select Monte Carlo photons.
- int **tel_select_mc_phot** (IO_BUFFER ∗iobuf)
- int **tel_select_mc_phot3d** (IO_BUFFER ∗iobuf)
- void **add_selector** (double m1, double m2, double E1, double E2, int c)
- int **select_bunches** (struct bunch ∗bunches, int ∗nbunches, double ∗photons)
- int **select_bunches3d** (struct bunch3d ∗bunches, int ∗nbunches, double ∗photons)
- void **syntax** (void)

Variables

- static int **interrupted** = 0
- static int **verbose** = 0
- struct bunch ∗ **tel_bunches** [MAXTEL]
- struct bunch3d ∗ **tel_bunches3d** [MAXTEL]
- int **max_bunches** [MAXTEL]
- int **max_bunches3d** [MAXTEL]
- int **tel_nbunches** [MAXTEL]
- int **tel_nbunches3d** [MAXTEL]
- double **tel_photons** [MAXTEL]
- double **tel_photons3d** [MAXTEL]
- double **obslev** = 1835.e2
- double **zdet** [MAXTEL]
- struct bunch ∗ **tel_bunches** [MAXTEL]
- struct bunch3d ∗ **tel_bunches3d** [MAXTEL]
- int **max_bunches** [MAXTEL]
- int **max_bunches3d** [MAXTEL]
- int **tel_nbunches** [MAXTEL]
- int **tel_nbunches3d** [MAXTEL]
- double **tel_photons** [MAXTEL]
- double **tel_photons3d** [MAXTEL]
- Selector ∗ **selectors** = NULL
- size_t **nselect** = 0
- static int **verbose** = 0
- struct bunch ∗ **sel_bunch** = NULL
- struct bunch3d ∗ **sel_bunch3d** = NULL
- int **sel_max** = 0
- int **sel_max3d** = 0

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 main()

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

Main program.

Main program function of hessio2iactio.cc program.

Main program function of select_iact program.

5.6 The read_iact program

Functions

- int my_print_simtel_mc_phot (IO_BUFFER ∗iobuf)

  Print Monte Carlo photons and photo-electrons.
- void **syntax** (void)
- void **show_header** (IO_ITEM_HEADER ∗item_header)
- int main (int argc, char ∗∗argv)

  Main program.

5.6.1 Detailed Description

5.6.2 Function Documentation

5.6.2.1 main()

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

Main program.

Main program function of read_hess.c program.

References verbose.

5.7 The check_trgmask program

Functions

- int **main** (int argc, char ∗∗argv)

5.7.1 Detailed Description

5.8 The extract_hess program

Functions

- static void syntax (char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**

5.8.1 Detailed Description

5.8.2 Function Documentation

5.8.2.1 main()

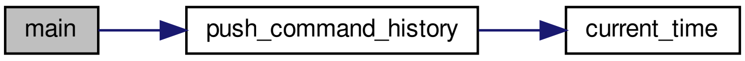
```
int main (
            int argc,
            char ** argv )
```

Main program.

Main program function of extract_hess.c program.

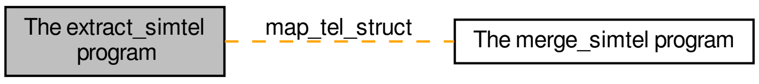
References push_command_history().

Here is the call graph for this function:



5.9 The extract_simtel program

Collaboration diagram for The extract_simtel program:



Data Structures

- struct map_tel_struct

  Structure with per output telescope information keeping track of prerequisites.

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void syntax (const char ∗program)

  Show program syntax.
- int find_in_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the input structures.
- int find_out_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the output structures.
- int find_mapped_telescope (int tel_id, int ifile)

  Mapping from telescope ID on input to telescope ID on output, with check.
- int write_io_block_to_file (IO_BUFFER ∗iobuf, FILE ∗f)

  Write an I/O block as-is to another file than foreseen for the I/O buffer.
- int **check_for_delayed_write** (IO_ITEM_HEADER ∗item_header, _unused_ int ifile, AllHessData ∗hsdata←_out, IO_BUFFER ∗iobuf_out)
- int merge_data_from_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header, int ifile, AllHessData ∗hsdata, AllHessData ∗hsdata_out, IO_BUFFER ∗iobuf_out)

  Processing of I/O blocks from the input file.
- int check_autoload_trgmask (const char ∗input_fname, IO_BUFFER ∗iobuf, int ifile)

  Check for a 'trgmask.gz' file matching the given input data file name and, if it exists, extract the corrected trigger bit patterns from it.
- void **print_process_status** (int prev_type1, int this_type1)
- int **read_map** (const char ∗map_fname)
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**
- static int **verbose** = 0
- struct map_tel_struct **map_tel** [H_MAX_TEL]
- int map_to [2][H_MAX_TEL+1]

  Mapping structures from input telescope ID to output telescope ID.
- int tel_idx [2][H_MAX_TEL+1]

  Mapping from telescope IDs to offsets in the data structures, first for input telescope IDs.
- int tel_idx_out [H_MAX_TEL+1]

  Mapping from output telescope ID to offset in output data structures.
- int **ntel1**
- int **ntel2**
- int **ntel**
- int **nrtel1**
- int **nrtel2**
- long **event1** = -1
- long **event2** = 0
- long **ev_hess_event** = 0
- long ev_pe_sum = 0

  For delayed writing.
- int **run1** = -1
- int **run2** = -1
- int **min_trg** = 2
- static struct trgmask_set ∗ **tms** [2] = { NULL, NULL }
- static struct trgmask_hash_set ∗ **ths** [2] = { NULL, NULL }
- static int **events** [2] = { 0, 0 }
- static int **mcshowers** [2] = { 0, 0 }
- static int **mcevents** [2] = { 0, 0 }
- static int **max_list** = 999

5.9.1 Detailed Description

5.9.2 Function Documentation

5.9.2.1 check_autoload_trgmask()

```
int check_autoload_trgmask (
             const char * input_fname,
             IO_BUFFER * iobuf,
             int ifile )
```

Check for a 'trgmask.gz' file matching the given input data file name and, if it exists, extract the corrected trigger bit patterns from it.

(Note: this is only relevant for multi-trigger data produced with a bug in recording the trigger bit pattern.)

We do not need to merge the contents of this file since the trigger bit patterns are corrected after reading the data.

5.9.2.2 stop_signal_function()

```
void stop_signal_function (
             int isig )
```

Stop the program gracefully when it catches an INT or TERM signal.

Parameters

isig	Signal number.

Returns

(none)

5.9.3 Variable Documentation

5.9.3.1 map_to

```
int map_to[2][H_MAX_TEL+1]
```

Mapping structures from input telescope ID to output telescope ID.

Not mapped telescopes are defined by output telescope ID of -1. The telescope ID to which a given input telescope ID should get mapped.

Referenced by find_mapped_telescope(), and find_out_tel_idx().

5.9.3.2 tel_idx

```
int tel_idx[2][H_MAX_TEL+1]
```

Mapping from telescope IDs to offsets in the data structures, first for input telescope IDs.

We restrict the ID/index mapping here to well behaved cases (0<ID<=H_MAX_TEL). An index value of -1 indicates a non-existant/ignored telescope. Where is a telescope of given ID in the input data structures?

Referenced by find_in_tel_idx(), and find_out_tel_idx().

5.9.3.3 tel_idx_out

```
int tel_idx_out[H_MAX_TEL+1]
```

Mapping from output telescope ID to offset in output data structures.

Where is a telescope of given ID in the output data structures?

Referenced by find_out_tel_idx().

5.10 The gen_trgmask program

Functions

- void **syntax** (char ∗prgname)
- int **main** (int argc, char ∗∗argv)

5.10.1 Detailed Description

5.11 The merge_simtel program

Collaboration diagram for The merge_simtel program:

The merge_simtel program
map_tel_struct
The extract_simtel program

Data Structures

- struct map_tel_struct
  Structure with per output telescope information keeping track of prerequisites.

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- int find_in_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the input structures.
- int find_out_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the output structures.
- int find_mapped_telescope (int tel_id, int ifile)

  Mapping from telescope ID on input to telescope ID on output, with check.
- int write_io_block_to_file (IO_BUFFER ∗iobuf, FILE ∗f)

  Write an I/O block as-is to another file than foreseen for the I/O buffer.
- int **has_min_trg_tel** (AllHessData ∗hsdata_out, int mtrg, double rtm)
- int check_for_delayed_write (IO_ITEM_HEADER ∗item_header, _unused_ int ifile, AllHessData ∗hsdata_out, IO_BUFFER ∗iobuf_out)

  Check if previously delayed writing of output should be done now.
- int merge_data_from_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header, int ifile, AllHessData ∗hsdata, AllHessData ∗hsdata_out, IO_BUFFER ∗iobuf_out)

  Processing and merging of I/O blocks from the two input files, hopefully presented in the right order.
- int check_autoload_trgmask (const char ∗input_fname, IO_BUFFER ∗iobuf, int ifile)

  Check for a 'trgmask.gz' file matching the given input data file name and, if it exists, extract the corrected trigger bit patterns from it.
- void **print_process_status** (int prev_type1, int this_type1, int prev_type2, int this_type2)
- int **read_map** (const char ∗map_fname)
- static void syntax (const char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**
- static int **verbose** = 0
- struct map_tel_struct **map_tel** [H_MAX_TEL]
- int map_to [2][H_MAX_TEL+1]

  Mapping structures from input telescope ID to output telescope ID.
- int tel_idx [2][H_MAX_TEL+1]

  Mapping from telescope IDs to offsets in the data structures, first for input telescope IDs.
- int tel_idx_out [H_MAX_TEL+1]

  Mapping from output telescope ID to offset in output data structures.
- int **ntel1**
- int **ntel2**
- int **ntel**
- int **nrtel1**
- int **nrtel2**
- long **event1** = -1
- long **event2** = 0
- long **ev_hess_event** = 0
- long ev_pe_sum = 0

  For delayed writing.
- int **run1** = -1
- int **run2** = -1

- int **min_trg** = 2
- double **distinct_sep** = 1.0
- static struct trgmask_set ∗ **tms** [2] = { NULL, NULL }
- static struct trgmask_hash_set ∗ **ths** [2] = { NULL, NULL }
- static int **events** [2] = { 0, 0 }
- static int **mcshowers** [2] = { 0, 0 }
- static int **mcevents** [2] = { 0, 0 }
- static int **max_list** = 999

5.11.1 Detailed Description

5.11.2 Function Documentation

5.11.2.1 check_autoload_trgmask()

```
int check_autoload_trgmask (
             const char * input_fname,
             IO_BUFFER * iobuf,
             int ifile )
```

Check for a 'trgmask.gz' file matching the given input data file name and, if it exists, extract the corrected trigger bit patterns from it.

(Note: this is only relevant for multi-trigger data produced with a bug in recording the trigger bit pattern.)

We do not need to merge the contents of this file since the trigger bit patterns are corrected after reading the data.

5.11.2.2 check_for_delayed_write()

```
int check_for_delayed_write (
             IO_ITEM_HEADER * item_header,
             _unused_ int ifile,
             AllHessData * hsdata_out,
             IO_BUFFER * iobuf_out )
```

Check if previously delayed writing of output should be done now.

Parameters

item_header	The item header descriptor.
ifile	Not used here since there is only one output file.
hsdata_out	The struct where the merged data gets collected.
iobuf_out	The output buffer descriptor.

5.11.2.3 stop_signal_function()

```
void stop_signal_function (
            int isig )
```

Stop the program gracefully when it catches an INT or TERM signal.

Parameters

isig	Signal number.

Returns

(none)

5.11.3 Variable Documentation

5.11.3.1 map_to

```
int map_to[2][H_MAX_TEL+1]
```

Mapping structures from input telescope ID to output telescope ID.

Not mapped telescopes are defined by output telescope ID of -1. The telescope ID to which a given input telescope ID should get mapped.

Referenced by find_mapped_telescope(), and find_out_tel_idx().

5.11.3.2 tel_idx

```
int tel_idx[2][H_MAX_TEL+1]
```

Mapping from telescope IDs to offsets in the data structures, first for input telescope IDs.

We restrict the ID/index mapping here to well behaved cases (0<ID<=H_MAX_TEL). An index value of -1 indicates a non-existant/ignored telescope. Where is a telescope of given ID in the input data structures?

Referenced by find_in_tel_idx(), and find_out_tel_idx().

5.11.3.3 tel_idx_out

```
int tel_idx_out[H_MAX_TEL+1]
```

Mapping from output telescope ID to offset in output data structures.

Where is a telescope of given ID in the output data structures?

Referenced by find_out_tel_idx().

5.12 The read_hess (aka read_simtel, read_cta) program

Data Structures

- struct next_file_struct
- struct range_list_struct

Macros

- #define CALIB_SCALE 0.92

  The factor needed to transform from mean p.e.

Typedefs

- typedef struct next_file_struct **NextFile**
- typedef struct range_list_struct **RangeList**

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void **init_rand** (int is)
- double grand48 (double mean, double sigma)

  Like RandFlat() from rndm2.c but using the drand48 engine.
- static void mc_event_fill (AllHessData ∗hsdata, double d_sp_idx)

  Fill histogram(s) for DST writing which require all MC shower and event data and which cannot be filled from DST level >= 2 data.
- static int write_dst_histos (IO_BUFFER ∗iobuf2)

  Write histograms for DST book-keeping and clear them afterwards.
- static void **show_run_summary** (AllHessData ∗hsdata, int nev, int ntrg, double plidx, double wsum_all, double wsum_trg, double rmax_x, double rmax_y, double rmax_r)
- static void syntax (char ∗program)

  Show program syntax.
- NextFile ∗ **add_next_file** (const char ∗fn, NextFile ∗nxt)
- RangeList ∗ **add_range** (long f, long t, RangeList ∗rl)
- int **is_in_range** (long n, RangeList ∗rl)
- int **read_disabled_pixels_list** (const char ∗fname, PixelDisabled ∗∗list)
- void show_header (IO_ITEM_HEADER ∗item_header)

  Print (to stdout) what information we have in the item header.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- struct basic_ntuple **bnt**
- static int **interrupted**
- static int **dst_processing**
- static int **g48_set**
- static double **g48_next**

5.12.1 Detailed Description

5.12.2 Macro Definition Documentation

5.12.2.1 CALIB_SCALE

```
#define CALIB_SCALE 0.92
```

The factor needed to transform from mean p.e.

units to units of the single-p.e. peak: Depends on the collection efficiency, the asymmetry of the single p.e. amplitude distribution and the electronic noise added to the signals.

5.12.3 Function Documentation

5.12.3.1 main()

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

Main program.

Main program function of read_hess.c program.

References verbose.

5.12.3.2 stop_signal_function()

```
void stop_signal_function (
             int isig )
```

Stop the program gracefully when it catches an INT or TERM signal.

Parameters

isig	Signal number.

Returns

(none)

5.13 The read_simtel_nr program

Macros

- #define **_UNUSED_**
- #define CALIB_SCALE 0.92

  The factor needed to transform from mean p.e.

Functions

- double calibrate_pixel_amplitude (AllHessData ∗hsdata, int itel, int ipix, int dummy, double cdummy)

  Calibrate a single pixel amplitude, for cameras with two gains per pixel.
- double **calibrate_pixel_amplitude** (AllHessData ∗hsdata, int itel, int ipix, _UNUSED_ int dummy, _↩ UNUSED_ double cdummy)
- int **user_get_type** (int itel)
- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void **show_run_summary** (AllHessData ∗hsdata, int nev, int ntrg, double plidx, double wsum_all, double wsum_trg, double rmax_x, double rmax_y, double rmax_r)
- static void syntax (char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**

5.13.1 Detailed Description

5.13.2 Macro Definition Documentation

5.13.2.1 CALIB_SCALE

```
#define CALIB_SCALE 0.92
```

The factor needed to transform from mean p.e.

units to units of the single-p.e. peak: Depends on the collection efficiency, the asymmetry of the single p.e. amplitude distribution and the electronic noise added to the signals.

5.13.3 Function Documentation

5.13.3.1 calibrate_pixel_amplitude()

```
double calibrate_pixel_amplitude (
             AllHessData * hsdata,
             int itel,
             int ipix,
             int dummy,
             double cdummy )
```

Calibrate a single pixel amplitude, for cameras with two gains per pixel.

This version does not include amplitude clipping nor obtaining amplitudes from the pixel timing data structure.

Returns

Pixel amplitude in peak p.e. units.

5.13.3.2 main()

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

Main program.

Main program function of read_hess.c program.

References verbose.

5.13.3.3 stop_signal_function()

```
void stop_signal_function (
             int isig )
```

Stop the program gracefully when it catches an INT or TERM signal.

Parameters

isig	Signal number.

Returns

(none)

5.14 The split_hessio program

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void syntax (char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**

5.14.1 Detailed Description

5.14.2 Function Documentation

5.14.2.1 main()

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

Main program.

Main program function of read_hess.c program.

References verbose.

5.14.2.2 stop_signal_function()

```
void stop_signal_function (
            int isig )
```

Stop the program gracefully when it catches an INT or TERM signal.

Parameters

isig	Signal number.

Returns

(none)

5.15 The hdata2hbook program (cvt2)

Functions

- int main (int argc, char ∗∗argv)
  Main program.

5.15.1 Detailed Description

5.16 The hdata2root program (cvt3)

Functions

- int **read_file** (IO_BUFFER ∗iobuf, const char ∗fname, int add_flag, int list_flag)
- int **main** (int argc, char ∗∗argv)

5.16.1 Detailed Description

Chapter 6

Data Structure Documentation

6.1 atmospheric_profile Struct Reference

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

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

Data Fields

- int atmprof_id

  Profile ID number ('atmprof<i>.dat') or 99.
- char ∗ atmprof_fname

  Original name of atmospheric profile loaded.
- double obslev

  Observation level [cm], a.s.l., as used in CORSIKA.
- unsigned n_alt

  Number of altitude levels.
- double ∗ alt_km

  Altitude a.s.l.
- double ∗ rho

  Density [g/cm^3] at each level.
- double ∗ thick

  Vertical column density from space to given level [g/cm^2].
- double ∗ refidx_m1

  Index of refraction minus one (n-1) at given level.
- int have_lay5_param

  Is 1 if the 5-layer CORSIKA built-in parametrization is known, 0 if not.
- double hlay [6]

  Layer bounderies a.s.l.
- double aatm [5]

  See ATMA CORSIKA inputs card.
- 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)
- double thickl [6]

  Atmospheric thickness at given hlay heights.
- double htoa

  Height (a.s.l.) at top of atmosphere [cm].

6.1.1 Detailed Description

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

6.1.2 Field Documentation

6.1.2.1 alt_km

```
double∗ atmospheric_profile::alt_km
```

Altitude a.s.l.

[km] at each level

Referenced by set_common_atmprof(), show_atmprof(), and write_atmprof().

6.1.2.2 hlay

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

Layer bounderies a.s.l.

[cm]; see ATMLAY CORSIKA inputs card

Referenced by atmegs_(), rhofc(), set_common_atmprof(), and show_atmprof().

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

- mc_atmprof.h

6.2 basic_ntuple Struct Reference

A struct with basic per-shower parameters, to be used as an n-tuple in the event selection.

```
#include <basic_ntuple.h>
```

Data Fields

- int primary

  Primary particle ID.
- int run

  Simulation run number.
- int event

  Event number (100∗shower number + array number)
- double weight

  Event weight, not to be used for selection (based on true energy).
- double lg_e_true

  log10(true energy of primary).
- double xfirst_true

  Atmospheric depth of first interaction.
- double xmax_true

  True shower maximum atmospheric depth (not well defined with few particles).
- double xc_true

  True core position at detection level (x coordinate).
- double yc_true

  True core position at detection level (y coordinate).
- double az_true

  True shower direction (Azimuth).
- double alt_true

  True shower direction (Altitude).
- double xc

  Reconstructed core position at detection level (x coordinate).
- double yc

  Reconstructed core position at detection level (y coordinate).
- double az

  Reconstructed shower direction (Azimuth).
- double alt

  Reconstructed shower direction (Altitude).
- double rcm

  Mean core distance of telescopes used in reconstruction.
- double mdisp

  Mean DISP (1.
- double theta

  Angle between source position and rec.
- double sig_theta

  R.m.s.
- double mscrw

  Mean scaled reduced width.
- double sig_mscrw

  R.m.s.
- double mscrl

  Mean scaled reduced length.
- double sig_mscrl

  R.m.s.
- double xmax

  Depth of shower maximum.
- double sig_xmax

R.m.s.

- double lg_e

  Log10 of reconstructed energy.
- double sig_e

  Relative error estimate on E (NOT the r.m.s.
- double chi2_e

  Consistency of individual energy estimates as reduced chi∗∗2 value.
- double tslope

  Core distance corrected mean time slope (deg/ns/100 m).
- double tsphere

  R.m.s.
- size_t n_img

  Number of used images.
- size_t n_trg

  Number of triggered telescopes.
- size_t n_fail

  Number of failed triggers (telescopes expected to trigger).
- size_t n_tsl0

  Number of images with zero time slope well outside light pool.
- size_t n_pix

  Total number of used pixels in all used images.
- size_t acceptance

  Event acceptance level by standard selection scheme (0: no; 1: shape cuts; 2: +angular cut; 3: +dE cut; 4: +dE2 cut; 5: +Hmax cut.

6.2.1 Detailed Description

A struct with basic per-shower parameters, to be used as an n-tuple in the event selection.

6.2.2 Field Documentation

6.2.2.1 mdisp

```
double basic_ntuple::mdisp
```

Mean DISP (1.

-width/length) of usable images.

6.2.2.2 sig_e

```
double basic_ntuple::sig_e
```

Relative error estimate on E (NOT the r.m.s.

of individual estimates).

6.2.2.3 sig_mscrl

```
double basic_ntuple::sig_mscrl
```

R.m.s.

of scaled reduced lengths of indvidual images.

6.2.2.4 sig_mscrw

```
double basic_ntuple::sig_mscrw
```

R.m.s.

of scaled reduced widths of individual images.

6.2.2.5 sig_theta

```
double basic_ntuple::sig_theta
```

R.m.s.

of theta of telescopes pairs (if $>$ 2 tel.).

6.2.2.6 sig_xmax

```
double basic_ntuple::sig_xmax
```

R.m.s.

of Xmax from individual telescopes/images.

6.2.2.7 theta

```
double basic_ntuple::theta
```

Angle between source position and rec.

shower direction.

6.2.2.8 tsphere

```
double basic_ntuple::tsphere
```

R.m.s.

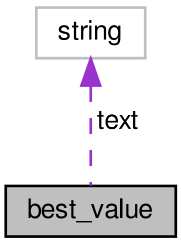
of trigger times from spherical propagation from shower max.

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

- basic_ntuple.h

6.3 best_value Struct Reference

Collaboration diagram for best_value:



Public Member Functions

- **best_value** (int k, double v, int qtr, const string &t, double aeff, double vlgE, double vlgE1, double vlgE2, double vds, double vbr=0., double vgr=0., double var=0., double ver=0., double veb=0., double ng=0., double nb=0.)

Data Fields

- int **kbin**
- double **best**
- int **q**
- string **text**
- double A

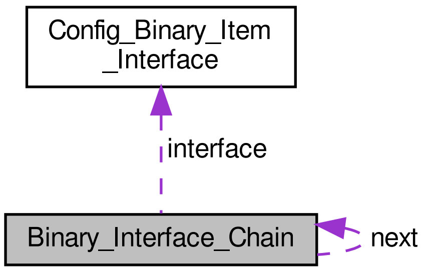
  effective area (for gammas)
- double **lgE**
- double **lgE1**
- double **lgE2**
- double **diff_sens**
- double **bg_rate**
- double **gamma_rate**
- double **angres**
- double **eres**
- double **ebias**
- double **n_gamma_cu**
- double **nint_gamma_cu**
- double **n_bg**
- double **nint_bg**

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

- best_of.cc

6.4 Binary_Interface_Chain Struct Reference

Collaboration diagram for Binary_Interface_Chain:



Data Fields

- struct Config_Binary_Item_Interface ∗ **interface**
- struct Binary_Interface_Chain ∗ **next**

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

- hconfig.c

6.5 bunch Struct Reference

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

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

Data Fields

- float photons

  Number of photons in bunch.
- float **x**
- float y

  Arrival position relative to telescope (cm)
- float **cx**
- float cy

  Direction cosines of photon direction.
- float ctime

  Arrival time (ns)
- float zem

  Height of emission point above sea level (cm)
- float lambda

  Wavelength in nanometers or 0.

6.5.1 Detailed Description

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

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

6.6 bunch3d Struct Reference

A more complete, alternative bunch structure which can also represent upward-going photon bunches or horizontal ones while the bunch and cbunch structures strictly assume downward-going photon bunches.

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

Data Fields

- float photons

  Number of photons in bunch.
- float **x**
- float **y**
- float z

  Arrival position relative to telescope (cm),.
- float **cx**
- float **cy**
- float cz

  Direction cosines of photon direction,.
- float ctime

  usually with cz $<$ 0 for downward.
- float dist

  Distance of emission point from arrival position (cm)
- float lambda

  Wavelength in nanometers or 0.

6.6.1 Detailed Description

A more complete, alternative bunch structure which can also represent upward-going photon bunches or horizontal ones while the bunch and cbunch structures strictly assume downward-going photon bunches.

While the coordinates of the sphere is still needed as extra information, for example its height for evaluating atmo-spheric extinction, the height of emission was replaced by the distance between emission and arrival position - no matter where this supposed arrival position is w.r.t. the fiducial sphere.

6.6.2 Field Documentation

6.6.2.1 ctime

```
float bunch3d::ctime
```

usually with cz < 0 for downward.

Arrival time (ns)

6.6.2.2 z

```
float bunch3d::z
```

Arrival position relative to telescope (cm),.

with the fiducial sphere center at (0,0,0).

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

- mc_tel.h

6.7 camera_nb_list Struct Reference

Data Fields

- int npix

    Number of pixels in camera.
- int nbsize

    Number of neighbours in list (elements in nblist).
- int ∗ pix_num_nb

    Number of neighbours for each pixel.
- int ∗ pix_first_nb

    Where in list is the first of the neighbours for each pixel.
- int ∗ nblist

    The actual packed list of all neighbours for all pixels.

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

- reconstruct.c

6.8 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 photons

  ph∗100
- short **x**
- short y

  x,y∗10 (mm)
- short **cx**
- short cy

  cx,cy∗30000
- short ctime

  ctime∗10 (0.1ns) after subtracting offset
- short log_zem

  log10(zem)∗1000
- short lambda

  (nm) or 0

6.8.1 Detailed Description

The compact_bunch struct is equivalent to the bunch struct except that we try to use less memory.

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 detecor 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

6.9 Config_Binary_Item_Interface Struct Reference

Interface definitions for binary-only items.

```
#include <hconfig.h>
```

Data Fields

- int io_item_type

  The eventio item type.
- int elem_size

  The size of the elements.
- void ∗(∗ new_func)(int nelem, int item_type)

  The function to be called for allocating elements.
- int(∗ delete_func)(void ∗ptr, int nelem, int item_type)

  The function to be called for deleting elements.
- int(∗ read_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type)

  The function to be called for reading elements from buffer.
- int(∗ write_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type)

  The function to be called for writing elements to buffer.
- int(∗ readtext_func)(void ∗bin_item, char ∗text, int item_type)

  The function to be called for reading elements from text line.
- int(∗ list_func)(void ∗bin_item, int item_type)

  The optional function for listing element contents.
- int(∗ copy_func)(void ∗bin_item_to, void ∗bin_item_from, int io_type)

  The optional function for copying elements.

6.9.1 Detailed Description

Interface definitions for binary-only items.

Binary-only items are structures, classes, or unions which can only be filled via dedicated functions (methods) and not via the standard text-input.

This structure defines available interface methods. The item type is always passed to the functions, in case that a function can handle more than one type.

6.9.2 Field Documentation

6.9.2.1 copy_func

```
int(* Config_Binary_Item_Interface::copy_func) (void *bin_item_to, void *bin_item_from, int
io_type)
```

The optional function for copying elements.

This is only needed if the element includes pointers to external or dynamically allocated material.

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

- hconfig.h

6.10 config_specific_data Struct Reference

Data Fields

- char **default_section** [65]

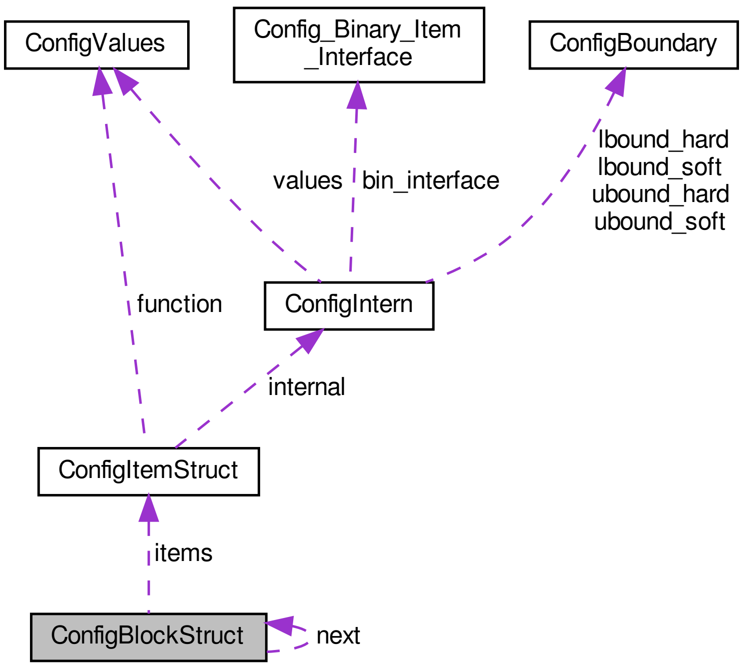
The documentation for this struct was generated from the following file:

- hconfig.c

6.11 ConfigBlockStruct Struct Reference

Configuration is organized in sections.

Collaboration diagram for ConfigBlockStruct:



Data Fields

- const char ∗ **section**
- struct ConfigItemStruct ∗ **items**
- struct ConfigBlockStruct ∗ **next**
- int **flag**

6.11.1 Detailed Description

Configuration is organized in sections.

CONFIG_BLOCK used for bookkeeping of that.

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

- hconfig.c

6.12 ConfigBoundary Union Reference

Configuration value may have optional lower and/or upper bounds.

```
#include <hconfig.h>
```

Data Fields

- long **lval**
- unsigned long **ulval**
- double ∗ **rval**

6.12.1 Detailed Description

Configuration value may have optional lower and/or upper bounds.

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

- hconfig.h

6.13 ConfigDataPointer Union Reference

This union of pointers allows convenient access of various types of data.

```
#include <hconfig.h>
```

Data Fields

- void ∗ **anything**
- char ∗ **cdata**
- unsigned char ∗ **ucdata**
- short ∗ **sdata**
- unsigned short ∗ **usdata**
- int ∗ **idata**
- unsigned int ∗ **uidata**
- long ∗ **ldata**
- unsigned long ∗ **uldata**
- float ∗ **fdata**
- double ∗ **ddata**
- bool ∗ **bdata**

6.13.1 Detailed Description

This union of pointers allows convenient access of various types of data.

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

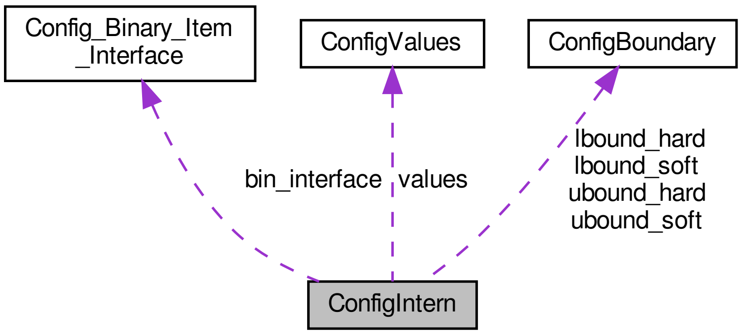
- hconfig.h

6.14 ConfigIntern Struct Reference

Configuration elements used only internally.

`#include <hconfig.h>`

Collaboration diagram for ConfigIntern:



Data Fields

- int itype

  Parameter type code.
- int elem_size

  Size of elements in bytes.
- int locked

  Set to 1 if locked.
- int bound

  Bits 0-3 set if lower soft, upper soft,

- int rcount

  Reconfiguration count.
- union ConfigBoundary lbound_soft

  Used for checking new values.
- union ConfigBoundary ubound_soft

  Used for checking new values.

- union ConfigBoundary lbound_hard

  Used for checking new values.
- union ConfigBoundary ubound_hard

  Used for checking new values.
- struct ConfigValues values

  Passed to user function.
- struct Config_Binary_Item_Interface ∗ **bin_interface**
- int **bin_alloc_elements**

6.14.1 Detailed Description

Configuration elements used only internally.

6.14.2 Field Documentation

6.14.2.1 bound

```
int ConfigIntern::bound
```

Bits 0-3 set if lower soft, upper soft,

lower hard, or upper hard bound present.

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

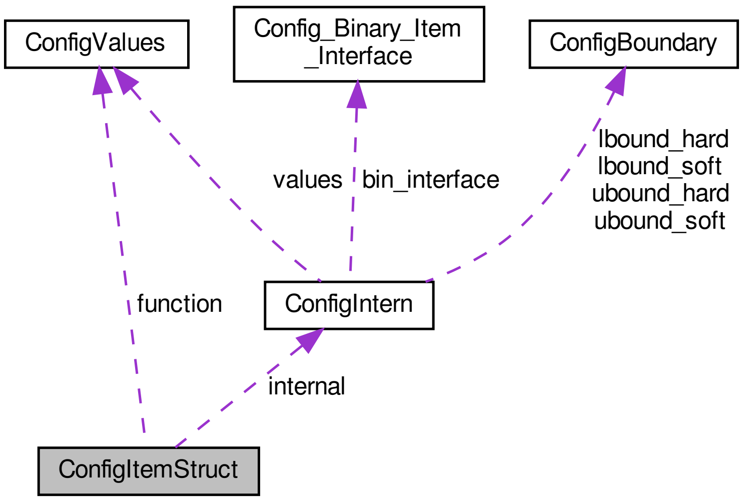
- hconfig.h

6.15 ConfigItemStruct Struct Reference

Configuration as used in definitions of configuration blocks.

```
#include <hconfig.h>
```

Collaboration diagram for ConfigItemStruct:



Data Fields

- const char ∗ name

  Parameter/function name.
- const char ∗ type

  Data/function type.
- int size

  Number of elements.
- void ∗ data

  Data pointer or NULL.
- PFIX function

  Associated function or NULL.
- const char ∗ initial

  Initial values/argument or NULL.
- const char ∗ lbound

  Lower bound (soft,hard) on values or NULL.
- const char ∗ ubound

  Upper bound (soft,hard) on values or NULL.
- int flags

  Additional flag bits.
- PFISS validate

  Function to validate if change is possible or NULL.
- void ∗ res1

  Placeholder to keep structure size the same.
- void ∗ res2

  Not used.
- struct ConfigIntern internal

  Internal data.

6.15.1 Detailed Description

Configuration as used in definitions of configuration blocks.

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

- hconfig.h

6.16 ConfigValues Struct Reference

Configuration values and supporting data passed to user functions.

```
#include <hconfig.h>
```

Data Fields

- void ∗ data_changed

    Pointer to the updated values.
- void ∗ data_saved

    Pointer to the saved values.
- int max_mod

    How many elements can, at most, be modified.
- int nmod

    How many have been modified.
- int ∗ list_mod

    List of indices to modified elements.
- unsigned char ∗ mod_flag

    Vector of size max_mod indicating modified elements.
- int itype

    Internal item type representation.
- const char ∗ name

    The name of the element.
- const char ∗ section

    The section to which it belongs.
- int elements

    The number of elements it has.
- int elem_size

    The size of one element in bytes.
- int binary_config

    Set to one if binary configuration was used.

6.16.1 Detailed Description

Configuration values and supporting data passed to user functions.

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

- hconfig.h

6.17 ebias_cor_data Struct Reference

Data Fields

- int **ndat**
- double ∗ **lgE**
- double ∗ **lgDE**

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

- user_analysis.c

6.18 ev_reg_chain Struct Reference

Use a double-linked list for the registry.

Collaboration diagram for ev_reg_chain:

ev_reg_chain
next
prev

Data Fields

- struct ev_reg_entry ∗ entry

  The current entry.
- struct ev_reg_chain ∗ **prev**
- struct ev_reg_chain ∗ **next**

6.18.1 Detailed Description

Use a double-linked list for the registry.

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

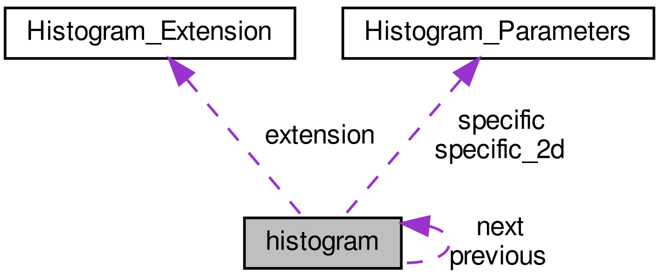
- eventio_registry.c

6.19 histogram Struct Reference

A complete 1-D or 2-D histogram with control and data elements.

```
#include <histogram.h>
```

Collaboration diagram for histogram:



Data Fields

- char ∗ title

  Histogram title (optional)
- long ident

  Histogram ID number (optional)
- union Histogram_Parameters **specific**
- union Histogram_Parameters **specific_2d**
- int nbins

  Number of histogram bins
- int nbins_2d

  Same for 2nd coordinate of 2-D.
- unsigned long entries

  No.
- unsigned long tentries

  No.
- unsigned long underflow

  No.
- unsigned long underflow_2d

  Same in 2nd coord of 2-D histo.
- unsigned long overflow

  No.
- unsigned long overflow_2d

  Same in 2nd coord of 2-D histo.
- unsigned long ∗ counts

Pointer to histogram data

- char type

  'I' for integer histogram,

- struct histogram ∗ previous

  References to neighbours in

- struct histogram ∗ next

  linked list of histograms.

- struct Histogram_Extension ∗ extension

  Extension for weighted histos

6.19.1 Detailed Description

A complete 1-D or 2-D histogram with control and data elements.

6.19.2 Field Documentation

6.19.2.1 entries

```
unsigned long histogram::entries
```

No.

of entries, incl. u.f./o.f.

Referenced by histogram_to_root(), print_ratio(), and write_dst_histos().

6.19.2.2 next

```
struct histogram* histogram::next
```

linked list of histograms.

Referenced by convert_histograms_to_root(), and write_histograms().

6.19.2.3 overflow

```
unsigned long histogram::overflow
```

No.

of entries above range

Referenced by histogram_to_root(), and print_ratio().

6.19.2.4 tentries

```
unsigned long histogram::tentries
```

No.

of entries, without """

Referenced by display_histogram(), fast_stat_histogram(), lookup_int(), lookup_real(), print_histogram(), print_↩ histogram_scaled(), and print_ratio().

6.19.2.5 type

```
char histogram::type
```

'I' for integer histogram,

'i' for int. lookup table,
'R' for floating point histogr. 'r' for fl. p. lookup table,
'F'/'D' for single/double pre- cision weighted histograms.

Referenced by aux_alloc_histogram(), display_2d_histogram(), display_histogram(), fast_stat_histogram(), fill_2d↩ _int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_histogram(), fill_int_histogram(), fill_↩ real_histogram(), fill_weighted_histogram(), histogram_matching(), histogram_to_root(), lookup_int(), lookup_real(), print_histogram(), print_histogram_scaled(), print_ratio(), and project_histogram().

6.19.2.6 underflow

```
unsigned long histogram::underflow
```

No.

of entries below range

Referenced by histogram_to_root(), and print_ratio().

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

- histogram.h

6.20 Histogram_Extension Struct Reference

A histogram extension only allocated for weighted histograms.

```
#include <histogram.h>
```

Data Fields

- double content_all

  Sum of all contents.
- double content_inside

  Sum of contents within range.
- double content_outside [8]

  Contents outside range.
- float ∗ fdata

  Data of each bin (ix+nx∗iy)
- double ∗ ddata

  in one of two precisions.

6.20.1 Detailed Description

A histogram extension only allocated for weighted histograms.

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

- histogram.h

6.21 Histogram_Parameters Union Reference

Parameters defining the usable range of coordinates.

```
#include <histogram.h>
```

Data Fields

- struct {
    double lower_limit
      Lower limit of histogram range.
    double upper_limit
      Upper limit of histogram range.
    double sum
      Sum of all values
    double tsum
      Sum of values within range
    double inverse_binwidth
      1.
  } real

Histogram parameters if it is some sort of 'F' or 'D' type.

- struct {
  long lower_limit
  Lower limit of histogram range.
  long upper_limit
  Upper limit of histogram range.
  long sum
  Sum of all values
  long tsum
  Sum of values within range
  long width
  Width of histogram range
  } integer

  Histogram parameters if it is some sort of 'I' (int) type.

6.21.1 Detailed Description

Parameters defining the usable range of coordinates.

6.21.2 Field Documentation

6.21.2.1

```
struct { ...  } Histogram_Parameters::integer
```

Histogram parameters if it is some sort of 'I' (int) type.

Needed for integer-type limits.

Referenced by histogram_matching(), histogram_to_root(), lookup_int(), print_histogram(), and print_ratio().

6.21.2.2 inverse_binwidth

```
double Histogram_Parameters::inverse_binwidth
```

1.

/(width_of_one_bin)

Referenced by lookup_real().

6.21.2.3

```
struct { ... } Histogram_Parameters::real
```

Histogram parameters if it is some sort of 'F' or 'D' type.

Needed for real-type limits.

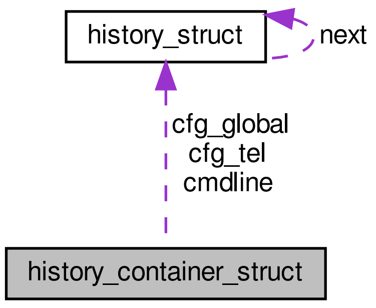
Referenced by histogram_matching(), histogram_to_root(), lookup_real(), print_histogram(), print_histogram_↩ scaled(), print_ratio(), and project_histogram().

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

- histogram.h

6.22 history_container_struct Struct Reference

Collaboration diagram for history_container_struct:



Data Fields

- int id

  Has always been 1.
- HSTRUCT ∗ cmdline

  Prior commands executed for current data.
- HSTRUCT ∗ cfg_global

  Global (or rather: not telescope-specific) configuration.
- HSTRUCT ∗∗ cfg_tel

  One linked list per telescope, if recognized.
- size_t ntel

  The number of telescopes for which cfg_tel was accumulated.

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

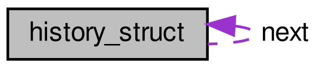
- io_history.h

6.23 history_struct Struct Reference

Use to build a linked list of configuration history.

```
#include <io_history.h>
```

Collaboration diagram for history_struct:



Data Fields

- char ∗ **text**
- time_t time

    Configuration test.
- struct history_struct ∗ next

    Time when the configuration was entered.

6.23.1 Detailed Description

Use to build a linked list of configuration history.

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

- io_history.h

6.24 histstat Struct Reference

Statistics element for histogram analysis.

```
#include <histogram.h>
```

Data Fields

- double **mean**
- double **mean_2d**
- double **tmean**
- double **tmean_2d**
- double **hmean**
- double **hmean_2d**
- double **sigma**
- double **sigma_2d**
- double **median**
- double **median_2d**

6.24.1 Detailed Description

Statistics element for histogram analysis.

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

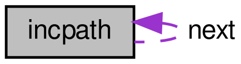
- histogram.h

6.25 incpath Struct Reference

An element in a linked list of include paths.

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

Collaboration diagram for incpath:



Data Fields

- char ∗ path

  The path name.
- struct incpath ∗ next

  The next element.

6.25.1 Detailed Description

An element in a linked list of include paths.

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

- fileopen.h

6.26 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

- char ∗ **text**
- struct linked_string ∗ **next**

6.26.1 Detailed Description

The linked_string is mainly used to keep CORSIKA input.

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

- mc_tel.h

6.27 map_tel_struct Struct Reference

Structure with per output telescope information keeping track of prerequisites.

Data Fields

- int tel_id

  Telescope ID on output.
- int ifn

  Input file number (1 only in this program)
- int inp_id

  Telescope ID on input.
- int inp_itel

  Sequential telescope count on input.
- int have_camset

  Have camera_settings for this telescope.
- int have_camorg

Have camera organisation for this telescope.
- int have_pixset

  Have pixel settings for this telescope.
- int have_pixdis

  Have pixels disabled for this telescope (optional)
- int have_camsoft

  Have camera software settings for this telescope.
- int have_pointcor

  Have pointing correction for this telescope.
- int have_trackset

  Have tracking settings for this telescope.

6.27.1 Detailed Description

Structure with per output telescope information keeping track of prerequisites.

6.27.2 Field Documentation

6.27.2.1 ifn

```
int map_tel_struct::ifn
```

Input file number (1 only in this program)

Input file number (1 or 2)

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

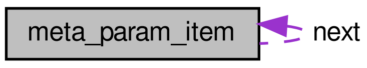
- extract_simtel.c
- merge_simtel.c

6.28 meta_param_item Struct Reference

A history meta parameter item consists of a parameter name and its text-formatted value, both as text strings.

```
#include <io_history.h>
```

Collaboration diagram for meta_param_item:



Data Fields

- char ∗ name

  Parameter name.
- char ∗ value

  Parameter value in text representation.
- struct meta_param_item ∗ next

  Pointer to next element in linked list or NULL for the last one.

6.28.1 Detailed Description

A history meta parameter item consists of a parameter name and its text-formatted value, both as text strings.

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

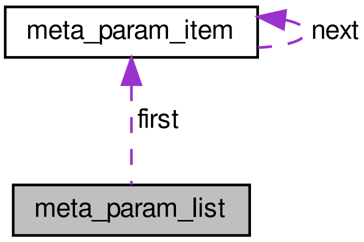
- io_history.h

6.29 meta_param_list Struct Reference

The linked MetaParamItem list for one ID (-1=global, detector ID otherwise) are registered under one list starting point.

```
#include <io_history.h>
```

Collaboration diagram for meta_param_list:



Data Fields

- long ident

  ID number (-1 for global parameter, 0 usually unused, >0 for detector ID)
- MetaParamItem ∗ first

  Pointer to start of the linked list or NULL.

6.29.1 Detailed Description

The linked MetaParamItem list for one ID (-1=global, detector ID otherwise) are registered under one list starting point.

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

- io_history.h

6.30 moments Struct Reference

Numbers to be summed up to obtain the moments.

```
#include <histogram.h>
```

Data Fields

- double **lower_limit**
- double **upper_limit**
- double **sum**
- double **tsum**
- double **sum2**
- double **tsum2**
- double **sum3**
- double **tsum3**
- double **sum4**
- double **tsum4**
- unsigned long **entries**
- unsigned long **tentries**
- int **level**

6.30.1 Detailed Description

Numbers to be summed up to obtain the moments.

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

- histogram.h

6.31 momstat Struct Reference

First, second, and higher moments of a 1-D histogram.

```
#include <histogram.h>
```

Data Fields

- double **mean**
- double **sigma**
- double **skewness**
- double **kurtosis**
- double **tmean**
- double **tsigma**
- double **tskewness**
- double **tkurtosis**

6.31.1 Detailed Description

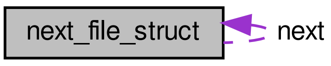
First, second, and higher moments of a 1-D histogram.

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

- histogram.h

6.32 next_file_struct Struct Reference

Collaboration diagram for next_file_struct:



Data Fields

- char ∗ **fname**
- struct next_file_struct ∗ **next**

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

- read_hess.c

6.33 photo_electron Struct Reference

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

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

Data Fields

- int pixel

  The pixel that was hit.
- int lambda

  The wavelength of the photon.
- double atime

  The time [ns] when the photon hit the pixel.

6.33.1 Detailed Description

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

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

- mc_tel.h

6.34 primary_id_struct Struct Reference

Data Fields

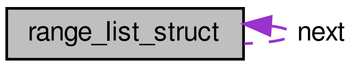
- int **id**
- const char $*$ **name** [NUM_LANG]

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

- camera_image.c

6.35 range_list_struct Struct Reference

Collaboration diagram for range_list_struct:



Data Fields

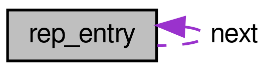
- long **from**
- long **to**
- struct range_list_struct ∗ **next**

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

- read_hess.c

6.36 rep_entry Struct Reference

Collaboration diagram for rep_entry:



Data Fields

- char ∗ **fname**
- char **mode** [4]
- size_t **count**
- struct rep_entry ∗ **next**

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

- fileopen.c

6.37 select_struct Struct Reference

Data Fields

- int **event**
- int **tel_id**
- int **pixel**
- int **gain**

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

- rh_dl0_test.c

6.38 selector Struct Reference

Data Fields

- double **min_mass**
- double **max_mass**
- double **min_energy**
- double **max_energy**
- int **charge**

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

- select_iact.c

6.39 shower_extra_parameters Struct Reference

Extra shower parameters of unspecified nature.

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

Data Fields

- long id

  May identify to the user what the parameters should mean.
- 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.
- double weight

  To be used if the weight of a shower may change during processing, e.g.
- size_t niparam

  Number of extra integer parameters.
- int ∗ iparam

  Space for extra integer parameters, at least of size niparam.
- size_t nfparam

  Number of extra floating-point parameters.
- float ∗ fparam

  Space for extra floats, at least of size nfparam.

6.39.1 Detailed Description

Extra shower parameters of unspecified nature.

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.

6.39.2 Field Documentation

6.39.2.1 weight

```
double shower_extra_parameters::weight
```

To be used if the weight of a shower may change during processing, e.g.

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:

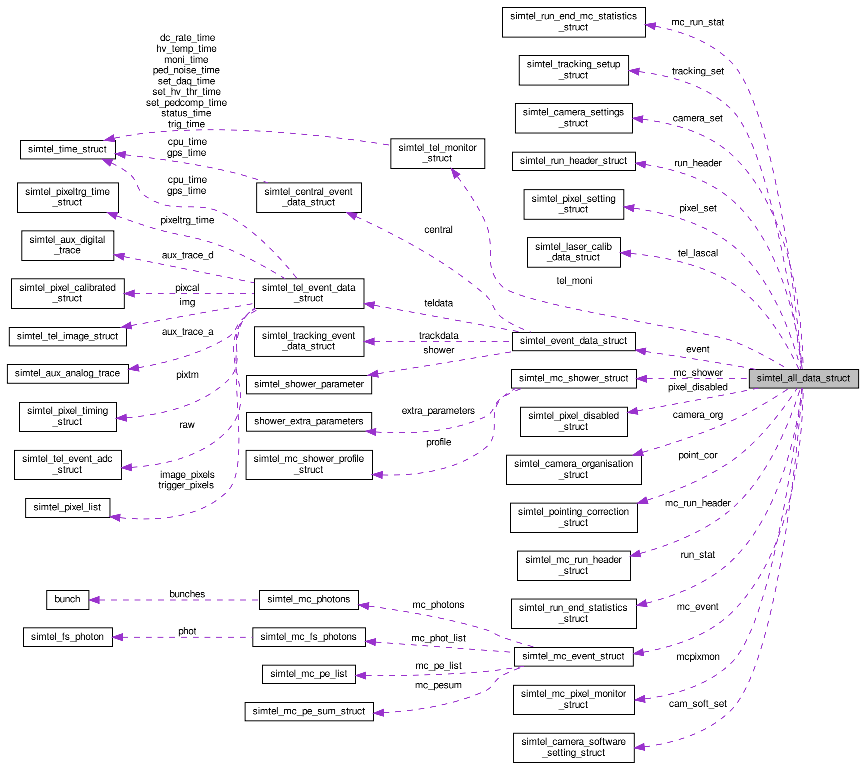
- mc_tel.h

6.40 simtel_all_data_struct Struct Reference

Container for all data.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_all_data_struct:



Data Fields

- RunHeader **run_header**
- MCRunHeader **mc_run_header**
- CameraSettings **camera_set** [H_MAX_TEL]
- CameraOrganisation **camera_org** [H_MAX_TEL]
- PixelSetting **pixel_set** [H_MAX_TEL]
- PixelDisabled **pixel_disabled** [H_MAX_TEL]
- CameraSoftSet **cam_soft_set** [H_MAX_TEL]
- TrackingSetup **tracking_set** [H_MAX_TEL]
- PointingCorrection **point_cor** [H_MAX_TEL]
- FullEvent **event**
- MCShower **mc_shower**
- MCEvent **mc_event**
- TelMoniData **tel_moni** [H_MAX_TEL]
- LasCalData **tel_lascal** [H_MAX_TEL]
- MCPixelMonitor **mcpixmon** [H_MAX_TEL]
- RunStat **run_stat**
- MCRunStat **mc_run_stat**

6.40.1 Detailed Description

Container for all data.

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

- io_hess.h

6.41 simtel_aux_analog_trace Struct Reference

Auxiliary analog trace (part of analog majority or sum trigger processing)

```
#include <io_hess.h>
```

Data Fields

- int known

  Must be set to 1 if and only if corresponding data is available.
- int tel_id

  Must match the expected telescope ID when reading.
- int trace_type

  Indicate what type of trace we have (1: pixel input, 2: analog sum, 3: disc/comp. output, 4: majority input)
- float time_scale

  Time per auxiliary sample over time per normal FADC sample (typ.: 0.25)
- size_t num_traces

  The number of traces coming from the camera.
- size_t len_traces

  The length of each trace in FADC samples.
- float ∗ trace_data

  Allocated on first use with num_traces∗len_traces elements.

6.41.1 Detailed Description

Auxiliary analog trace (part of analog majority or sum trigger processing)

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

- io_hess.h

6.42 simtel_aux_digital_trace Struct Reference

Auxiliary digital trace (derived from FADC samples)

```
#include <io_hess.h>
```

Data Fields

- int known
  Must be set to 1 if and only if corresponding data is available.
- int tel_id
  Must match the expected telescope ID when reading.
- int trace_type
  Indicate what type of trace we have (1: DigitalSum trigger trace)
- float time_scale
  Time per auxiliary sample over time per normal FADC sample (typ.: 1.0)
- size_t num_traces
  The number of traces coming from the camera.
- size_t len_traces
  The length of each trace in FADC samples.
- uint16_t ∗ trace_data
  Allocated on first use with num_traces∗len_traces elements.

6.42.1 Detailed Description

Auxiliary digital trace (derived from FADC samples)

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

- io_hess.h

6.43 simtel_camera_organisation_struct Struct Reference

Logical organisation of camera electronics channels.

```
#include <io_hess.h>
```

Data Fields

- int tel_id
  Telescope ID.
- int num_pixels
  Number of pixels in camera.
- int num_drawers
  Number of drawers (mechanical units, hardware modules) in camera.
- int num_gains
  Number of gains per PM.
- int num_sectors
  Number of sectors (trigger groups).
- int drawer [H_MAX_PIX]
  Drawer (hardware module) assignment for each pixel.
- int **card** [H_MAX_PIX][H_MAX_GAINS]
- int **chip** [H_MAX_PIX][H_MAX_GAINS]
- int **channel** [H_MAX_PIX][H_MAX_GAINS]
- int nsect [H_MAX_PIX]
  Number of sectors (trigger groups) for trigger(s).
- int sectors [H_MAX_PIX][H_MAX_PIXSECTORS]
  Pixels in sectors (trigger groups).
- int sector_type [H_MAX_SECTORS]
  0: majority, 1: analog sum, 2: digital sum, 3: digital majority
- double sector_threshold [H_MAX_SECTORS]
  Multiplicity or sum threshold applied to sector. [mV ?].
- double sector_pixthresh [H_MAX_SECTORS]
  Pixel threshold for majority or clipping limit for sum triggers. [mV ?].

6.43.1 Detailed Description

Logical organisation of camera electronics channels.

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

- io_hess.h

6.44 simtel_camera_settings_struct Struct Reference

Definition of camera optics settings.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

  Telescope ID.
- int num_pixels

  Number of pixels in camera.
- double xpix [H_MAX_PIX]

  Pixel x position in camera [m].
- double ypix [H_MAX_PIX]

  Pixel y position in camera [m].
- double zpix [H_MAX_PIX]

  Pixel z position w.r.t. focal plane in camera center [m]. {new}.
- double nxpix [H_MAX_PIX]

  Pixel pointing direction (nx,ny,1) x component. {new}.
- double nypix [H_MAX_PIX]

  Pixel pointing direction (nx,ny,1) y component. {new}.
- double area [H_MAX_PIX]

  Pixel active area ([m^ 2]).
- double size [H_MAX_PIX]

  Pixel diameter (flat-to-flat, [m]).
- int pixel_shape [H_MAX_PIX]

  Pixel shape type (0: circ., 1,3: hex, 2: square, -1: unknown). {new}.
- double cam_rot

  Rotation angle of camera (counter-clock-wise from back side for prime focus camera).
- double flen

  Focal length of optics (geometric or nominal) [m].
- double eff_flen

  Suggested effective focal length for image scale (can be zero). [m].
- double eff_flen_x

  Value may be different in x projection if mirror not rotationally symmetric. [m].
- double eff_flen_y

  Value may be different in y projection if mirror not rotationally symmetric. [m].
- double eff_flen_dx

  Displacement of image c.o.g. in x by asymmetric mirror [m].
- double eff_flen_dy

  Displacement of image c.o.g. in y by asymmetric mirror [m].
- int num_mirrors

  Number of mirror tiles.
- double mirror_area

  Total area of individual mirrors corrected for inclination [m^ 2].
- int curved_surface

  0 for flat surface, 1 for curved surface. {new}
- int pixels_parallel

  0 if (some) pixels are inclined, 1 if all pixels are parallel {new}
- int common_pixel_shape

  instead of individual pixel shape if al pixels are the same. {new}

6.44.1 Detailed Description

Definition of camera optics settings.

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

- io_hess.h

6.45 simtel_camera_software_setting_struct Struct Reference

Software settings used in camera process.

```
#include <io_hess.h>
```

Data Fields

- int tel_id
  The telescope ID number (1 ... n)
- int **dyn_trig_mode**
- int **dyn_trig_threshold**
- int **dyn_HV_mode**
- int **dyn_HV_threshold**
- int data_red_mode
  The desired data reduction mode.
- int zero_sup_mode
  The desired zero suppression mode.
- int zero_sup_num_thr
  The number of thresholds to be used by z.s.
- int zero_sup_thresholds [10]
  Threshold values to be used by z.s.
- int **unbiased_scale**
- int **dyn_ped_mode**
- int **dyn_ped_events**
- int dyn_ped_period
  [ms]
- int monitor_cur_period
  [ms]
- int report_cur_period
  [ms]
- int monitor_HV_period
  [ms]
- int report_HV_period
  [ms]

6.45.1 Detailed Description

Software settings used in camera process.

6.45.2 Field Documentation

6.45.2.1 zero_sup_mode

```
int simtel_camera_software_setting_struct::zero_sup_mode
```

The desired zero suppression mode.

The mode actually used may depend on the data.

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

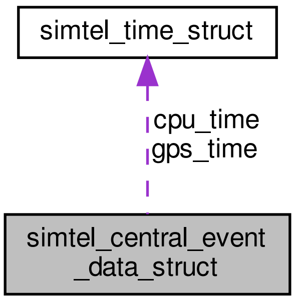
- io_hess.h

6.46 simtel_central_event_data_struct Struct Reference

Central trigger event data.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_central_event_data_struct:



Data Fields

- int glob_count

  Global event count.
- HTime cpu_time

  CPU time at central trigger station.
- HTime gps_time

  GPS time at central trigger station.
- int teltrg_pattern

  Bit pattern of telescopes having sent a trigger signal to the central station.
- int teldata_pattern

  Bit pattern of telescopes having sent event data that could be merged.
- int num_teltrg

  How many telescopes triggered.
- int teltrg_list [H_MAX_TEL]

  List of IDs of triggered telescopes.
- float teltrg_time [H_MAX_TEL]

  Relative time of trigger signal.
- int teltrg_type_mask [H_MAX_TEL]

  Bit mask which type of trigger fired.
- float teltrg_time_by_type [H_MAX_TEL][H_MAX_TRG_TYPES]

  Time of trigger separate for each type.
- int num_teldata

  Number of telescopes expected to have data.
- int teldata_list [H_MAX_TEL]

  List of IDs of telescopes expected to have data.
- double az_comp

  Azimuth angle for which plane wavefront compensation was evaluated. [radian].
- double alt_comp

  Altitude angle for which plane wavefront compensation was evaluated. [radian].
- double ls_comp

  Assumed light speed (in air) for plane wavefront compensation. [cm/ns].

6.46.1 Detailed Description

Central trigger event data.

6.46.2 Field Documentation

6.46.2.1 teldata_list

```
int simtel_central_event_data_struct::teldata_list[H_MAX_TEL]
```

List of IDs of telescopes expected to have data.

Keep in mind that due to telescope dead time etc., or that some processing step discarded that data, that is not a guarantee that actual data is available. Check in telescope data structure for actually available data.

6.46.2.2 teldata_pattern

```
int simtel_central_event_data_struct::teldata_pattern
```

Bit pattern of telescopes having sent event data that could be merged.

(Historical; only useful for small no. of telescopes.)

6.46.2.3 teltrg_pattern

```
int simtel_central_event_data_struct::teltrg_pattern
```

Bit pattern of telescopes having sent a trigger signal to the central station.

(Historical; only useful for small no. of telescopes.)

6.46.2.4 teltrg_time

```
float simtel_central_event_data_struct::teltrg_time[H_MAX_TEL]
```

Relative time of trigger signal.

after correction for nominal delay [ns].

6.46.2.5 teltrg_time_by_type

```
float simtel_central_event_data_struct::teltrg_time_by_type[H_MAX_TEL][H_MAX_TRG_TYPES]
```

Time of trigger separate for each type.

Check bits 0-3 of the corresponding teltrg_type_mask variable for trigger types fired in the camera. For cameras with only a single trigger type, just look at teltrg_time.

6.46.2.6 teltrg_type_mask

```
int simtel_central_event_data_struct::teltrg_type_mask[H_MAX_TEL]
```

Bit mask which type of trigger fired.

More than one trigger type per telescope is possible. Bits well beyond H_MAX_TRG_TYPES are modifier flags, in addition to trigger type(s). Bit 0: (mostly analog) majority trigger. Bit 1: analog sum trigger. Bit 2: digital sum trigger (working on readout data stream). Bit 3: digital (majority) trigger (working on separately digitized data). Bits 4-7: reserved (must be zero). Bit 8: long-event modifier (readout may include more samples than normal). Bit 9: matching muon-ring enhancement conditions, acceptable mono event. Bit 10: randomly chosen as acceptable mono event. Bits 11-15: unspecified (not used so far). Bits 16-31: reserved (must be zero).

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

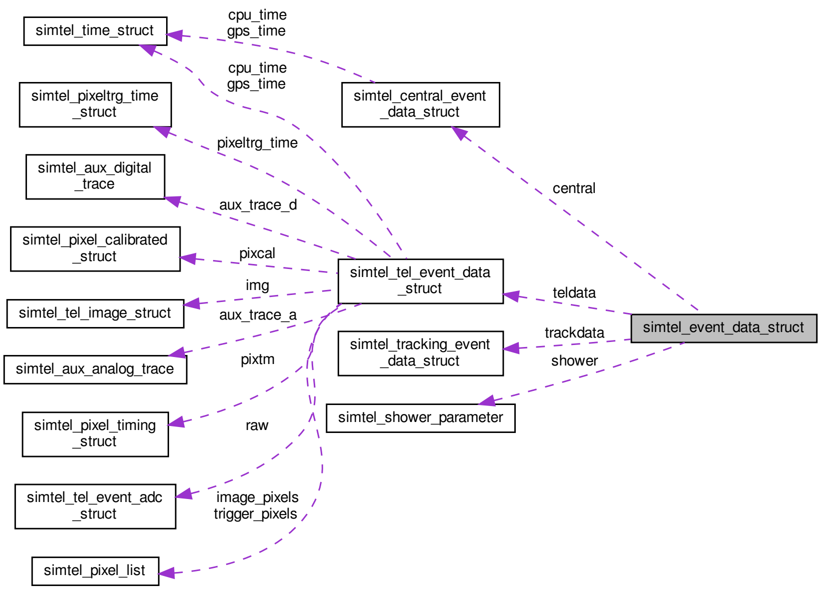
- io_hess.h

6.47 simtel_event_data_struct Struct Reference

All data for one event.

`#include <io_hess.h>`

Collaboration diagram for simtel_event_data_struct:



Data Fields

- int num_tel

  Number of telescopes in run.
- CentralEvent central

  Central trigger data and data pattern.
- TelEvent teldata [H_MAX_TEL]

  Raw and/or image data.
- TrackEvent trackdata [H_MAX_TEL]

  Interpolated tracking data.
- ShowerParameters shower

  Reconstructed shower parameters.
- int num_teldata

  Number of telescopes for which we actually have data.
- int teldata_list [H_MAX_TEL]

  List of IDs of telescopes with data.

6.47.1 Detailed Description

All data for one event.

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

- io_hess.h

6.48 simtel_fs_photon Struct Reference

Single photon incident on focal surface, after ray-tracing in telescope optics.

```
#include <io_hess.h>
```

Data Fields

- float **x**
- float y

  Impact position, projected [cm].
- float **cx**
- float cy

  Direction cosines w.r.t. focal surface normal.
- float prob

  Probability not accounted for yet.
- uint16_t wavelength

  Wavelength [nm].
- uint16_t flags

  ?

6.48.1 Detailed Description

Single photon incident on focal surface, after ray-tracing in telescope optics.

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

- io_hess.h

6.49 simtel_laser_calib_data_struct Struct Reference

Laser calibration data.

```
#include <io_hess.h>
```

Data Fields

- int known

  Are the calibration values known?
- int tel_id

  Telescope ID.
- int num_pixels

  Number of pixels.
- int num_gains

  Number of gains.
- int lascal_id

  Laser calibration ID.
- double calib [H_MAX_GAINS][H_MAX_PIX]

  ADC to laser/LED p.e.
- double max_int_frac [H_MAX_GAINS]

  Maximum fraction of the signal which can be in the fixed integration window.
- double max_pixtm_frac [H_MAX_GAINS]

  Maximum fraction of the signal which can be in the pixel timing integration.
- double tm_calib [H_MAX_GAINS][H_MAX_PIX]

  Transit time calibration [ns].
- double ff_corr [H_MAX_GAINS][H_MAX_PIX]

  Flat-field correction as part of 'calib'.

6.49.1 Detailed Description

Laser calibration data.

6.49.2 Field Documentation

6.49.2.1 calib

```
double simtel_laser_calib_data_struct::calib[H_MAX_GAINS][H_MAX_PIX]
```

ADC to laser/LED p.e.

conversion, in [mean p.e.], details depending on calibration procedure.

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

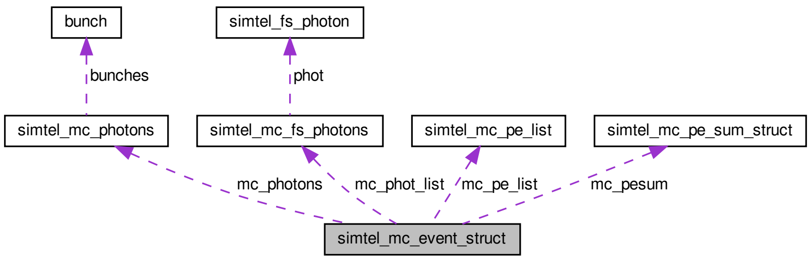
- io_hess.h

6.50 simtel_mc_event_struct Struct Reference

Monte Carlo event-specific data.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_mc_event_struct:



Data Fields

- int event

  Event number -> global counter.
- int shower_num

  Shower number as in shower structure.
- double xcore

  Core position w.r.t. array reference point [m],.
- double ycore

  x -> N, y -> W.
- double aweight

  *Area weight (units: [m**2]) in case of non-uniform sampling, normallly counted in the shower plane and normalized such that the sum over all events for a shower should, on average, be the area over which core offsets are thrown (see also num_use and core_range in MCRunHeader).*
- double photons [H_MAX_TEL]

  The CORSIKA photon sum into fiducial volume.
- MCpeSum mc_pesum

  Numbers of / sums of photo-electrons.
- MCphotons mc_photons [H_MAX_TEL]

  Raw simulated photons (fiducial sphere).
- MCpeList mc_pe_list [H_MAX_TEL]

  List of detected photo-electrons.
- MCfsPhotons mc_phot_list [H_MAX_TEL]

  List of photons imaged onto focal surface.

6.50.1 Detailed Description

Monte Carlo event-specific data.

6.50.2 Field Documentation

6.50.2.1 aweight

```
double simtel_mc_event_struct::aweight
```

Area weight (units: [m∗∗2]) in case of non-uniform sampling, normallly counted in the shower plane and normalized such that the sum over all events for a shower should, on average, be the area over which core offsets are thrown (see also num_use and core_range in MCRunHeader).

It may be zero for uniform sampling.

Referenced by mc_event_fill().

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

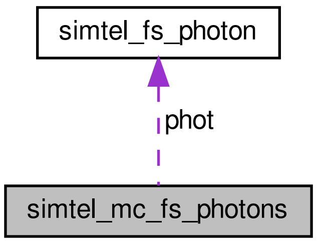
- io_hess.h

6.51 simtel_mc_fs_photons Struct Reference

List of photons incident on focal surface.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_mc_fs_photons:



Data Fields

- int nphot

    Number of photons to record.
- FSphoton ∗ phot

    Only allocated on demand; not needed for normal simulations.
- int max_phot

    How many we can store in 'phot' above, without re-allocating.

6.51.1 Detailed Description

List of photons incident on focal surface.

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

- io_hess.h

6.52 simtel_mc_pe_list Struct Reference

Photo-electrons registered in pixels all listed individually.

```
#include <io_hess.h>
```

Data Fields

- int npe

    The number of all photo-electrons in the telescope.
- int pixels

    The number of pixels in the camera.
- int flags

    Bit 0: with amplitudes, bit 1: includes NSB.
- int pe_count [H_MAX_PIX]

    The numbers of p.e. at each pixel.
- int itstart [H_MAX_PIX]

    The start index for each pixel in the sequential atimes vector.
- double ∗ atimes

    The list of start times of all photo-eletrons.
- double ∗ amplitudes

    Optional list of matching amplitudes [mean p.e.].
- int max_npe

    How many p.e. we can store in the atimes (+amplitudes) vector(s).

6.52.1 Detailed Description

Photo-electrons registered in pixels all listed individually.

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

- io_hess.h

6.53 simtel_mc_pe_sum_struct Struct Reference

Sums of photo-electrons in MC (total and per pixel).

```
#include <io_hess.h>
```

Data Fields

- int event

  Event number -> global counter.
- int shower_num

  Shower number as in shower structure.
- int num_tel

  Number of telescopes simulated.
- int num_pe [H_MAX_TEL]

  Number of photo-electrons per telescope.
- int num_pixels [H_MAX_TEL]

  Pixels per telescope or 0.
- int pix_pe [H_MAX_TEL][H_MAX_PIX]

  Photo-electrons per pixel (without NSB).
- double photons [H_MAX_TEL]

  The sum of the photon content of all bunches.
- double photons_atm [H_MAX_TEL]

  Photons surviving atmospheric transmission.
- double photons_atm_3_6 [H_MAX_TEL]

  Photons surv. atm. tr. in the 300 to 600 nm range.
- double photons_atm_400 [H_MAX_TEL]

  Photons surv. atm. tr. in the 350 to 450 nm range.
- double photons_atm_qe [H_MAX_TEL]

  Photons surviving atmospheric transmission, mirror reflectivity (except funnel), and Q.E.

6.53.1 Detailed Description

Sums of photo-electrons in MC (total and per pixel).

6.53.2 Field Documentation

6.53.2.1 photons_atm_qe

```
double simtel_mc_pe_sum_struct::photons_atm_qe[H_MAX_TEL]
```

Photons surviving atmospheric transmission, mirror reflectivity (except funnel), and Q.E.

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

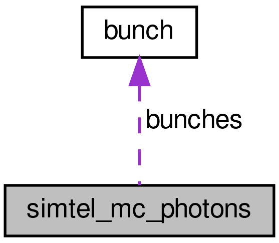
- io_hess.h

6.54 simtel_mc_photons Struct Reference

Collection of photons from Monte Carlo, as received from CORSIKA or LightEmission.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_mc_photons:



Data Fields

- struct bunch ∗ bunches

  Bunches of photons.
- int nbunches

  How many photon bunches we have at this telescope.
- int max_bunches

  How many we can store in 'bunches' vector above.
- double photons

  The sum of the photon content of all bunches.

6.54.1 Detailed Description

Collection of photons from Monte Carlo, as received from CORSIKA or LightEmission.

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

- io_hess.h

6.55 simtel_mc_pixel_monitor_struct Struct Reference

Monte Carlo pixel 'monitoring' with parameters as actually used in simulation.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

    Telescope ID number.
- int flags

    Bit 0: NSB p.e.
- int num_pixels

    Number of pixels in camera.
- int num_gains

    Number of different electronics gains for read-out (1 or 2).
- double nsb_pe_rate [H_MAX_PIX]

    NSB pixel p.e. rate [p.e./ns].
- double qe_rel [H_MAX_PIX]

    Assumed QE/PDE w.r.t. nominal.
- double gain_rel [H_MAX_PIX]

    Assumed (PMT/common) gain w.r.t. nominal.
- double hv_rel [H_MAX_PIX]

    Assumed high voltage w.r.t. nominal.
- double current [H_MAX_PIX]

    Assumed current at pixel [uA].
- double fadc_amp [H_MAX_GAINS][H_MAX_PIX]

    Assumed FADC amplitude per mean p.e.
- BYTE disabled [H_MAX_PIX]

    Pixel totally off and/or disabled in trigger.
- double delay [H_MAX_PIX]

    Assumed PMT transit time (HV dependent) plus other delays on signal path [ns].

6.55.1 Detailed Description

Monte Carlo pixel 'monitoring' with parameters as actually used in simulation.

6.55.2 Field Documentation

6.55.2.1 flags

```
int simtel_mc_pixel_monitor_struct::flags
```

Bit 0: NSB p.e.

rate, bit 1: rel. QE, bit 2: gain rel., bit 3: HV rel., bit 4: current, bit 5: fadc_amp (HG), bit 6: fadc_amp (LG, if applicable), bit 7: disabled status, bit 8: time delay (sum of sensor, signal path, etc.)

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

- io_hess.h

6.56 simtel_mc_run_header_struct Struct Reference

MC run header.

```
#include <io_hess.h>
```

Data Fields

- int shower_prog_id

  Recorded data:
- int shower_prog_vers

  version ∗ 1000
- time_t shower_prog_start

  Time when shower simulation of run started (CORSIKA: only date)
- int detector_prog_id

  sim_telarray=1, ...
- int detector_prog_vers

  version ∗ 1000
- time_t detector_prog_start

  Time when detector simulation of run started
- double obsheight

  Height of simulated observation level.
- int num_showers

  Number of showers (intended to be) simulated.
- int num_use

  Number of uses of each shower.
- int core_pos_mode

  Core position fixed/circular/rectangular/...
- double core_range [2]

  rmin+rmax or dx+dy [m].
- double az_range [2]

  Range of shower azimuth [rad, N->E].
- double alt_range [2]

  Range of shower altitude [rad].
- int diffuse

  Diffuse mode off/on.
- double viewcone [2]

  Min.+max. opening angle for diffuse mode [degrees] (was always in degrees despite earlier '[rad]' comment).
- double E_range [2]

  Energy range [TeV] of simulated showers.
- double spectral_index

  Power-law spectral index of spectrum (<0).
- double B_total

  Total geomagnetic field assumed [microT].
- double B_inclination

  Inclination of geomagnetic field [rad].
- double B_declination

  Declination of geomagnetic field [rad].
- double injection_height

Not used. See depth_start in MCShower instead. (Height of particle injection [m].)
- double fixed_int_depth

  Not used. See h_first_int in MCShower instead. (Fixed depth of first interaction or 0 [g/cm^2].)
- int atmosphere

  Atmospheric model number.
- int **corsika_iact_options**
- int **corsika_low_E_model**
- int **corsika_high_E_model**
- double **corsika_bunchsize**
- double **corsika_wlen_min**
- double **corsika_wlen_max**
- int **corsika_low_E_detail**
- int **corsika_high_E_detail**

6.56.1 Detailed Description

MC run header.

6.56.2 Field Documentation

6.56.2.1 shower_prog_id

```
int simtel_mc_run_header_struct::shower_prog_id
```

Recorded data:

CORSIKA=1, ALTAI=2, KASCADE=3, MOCCA=4.

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

- io_hess.h

6.57 simtel_mc_shower_profile_struct Struct Reference

Monte Carlo shower profile (sort of histogram).

```
#include <io_hess.h>
```

Data Fields

- int id

  Type of profile (also determines units below).
- int num_steps

  Number of histogram steps.
- int max_steps

  Number of allowed steps as allocated for content.
- double start

  Start of ordinate ([m] or [g/cm^2])
- double end

  End of it.
- double binsize

  (End-Start)/num_steps; not saved
- double ∗ content

  Histogram contents (allocated on demand).

6.57.1 Detailed Description

Monte Carlo shower profile (sort of histogram).

6.57.2 Field Documentation

6.57.2.1 id

```
int simtel_mc_shower_profile_struct::id
```

Type of profile (also determines units below).

```
Temptative definitions:
@li 1000*k + 1:  Profile of all charged particles.
@li 1000*k + 2:  Profile of electrons+positrons.
@li 1000*k + 3:  Profile of muons.
@li 1000*k + 4:  Profile of hadrons.
@li 1000*k + 10: Profile of Cherenkov photon emission [1/m].

The value of k specifies the binning:
@li k = 0: The profile is in terms of atmospheric depth
    along the shower axis.
@li k = 1: in terms of vertical atmospheric depth.
@li k = 2: in terms of altitude [m] above sea level.
```

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

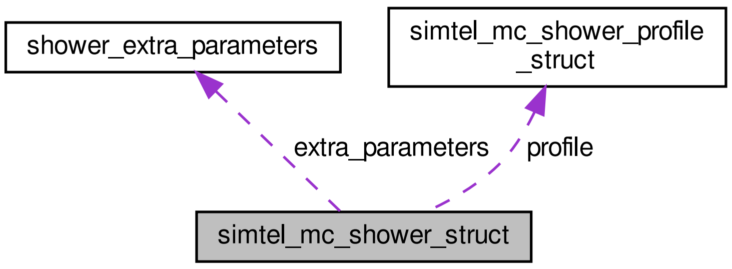
- io_hess.h

6.58 simtel_mc_shower_struct Struct Reference

Shower specific data.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_mc_shower_struct:



Data Fields

- int **shower_num**
- int primary_id

  Particle ID of primary.
- double energy

  primary energy [TeV]
- double azimuth

  Azimuth (N->E) [rad].
- double altitude

  Altitude [rad].
- double depth_start

  Atmospheric depth where particle started [g/cm^2].
- double h_first_int

  height of first interaction a.s.l. [m]
- double xmax

  Atmospheric depth of shower maximum [g/cm^2], derived from all charged particles.
- double hmax

  Height of shower maximum [m] in xmax.
- double emax

  Atm. depth of maximum in electron number.
- double cmax

  Atm. depth of max. in Cherenkov photon emission.
- int num_profiles

  Number of profiles filled.
- ShowerProfile **profile** [H_MAX_PROFILE]
- struct shower_extra_parameters **extra_parameters**

6.58.1 Detailed Description

Shower specific data.

6.58.2 Field Documentation

6.58.2.1 primary_id

```
int simtel_mc_shower_struct::primary_id
```

Particle ID of primary.

Was in CORSIKA convention where detector_prog_vers in MC run header was 0, and is now 0 (gamma), 1(e-), 2(mu-), 100∗A+Z for nucleons and nuclei, negative for antimatter.

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

- io_hess.h

6.59 simtel_pixel_calibrated_struct Struct Reference

Pixel signal intensities calibrated in some sort of p.e.

```
#include <io_hess.h>
```

Data Fields

- int known

  is calibrated pixel data known?
- int tel_id

  Telescope ID.
- int num_pixels

  Pixels in camera: list should be in this range.
- int int_method

  -2 (timing local peak), -1 (timing global peak), >=0 (integration scheme, if known)
- int list_known

  Was list of significant pixels filled in? 1: use list, 2: all pixels significant.
- int list_size

  Size of the list of available pixels (with list mode).
- int pixel_list [H_MAX_PIX]

  List of available pixels (with list mode).
- uint8_t significant [H_MAX_PIX]

  Was amplitude large enough to record it?
- float pixel_pe [H_MAX_PIX]

  Calibrated & flat-fielded pixel intensity [p.e.].

6.59.1 Detailed Description

Pixel signal intensities calibrated in some sort of p.e.

scale

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

- io_hess.h

6.60 simtel_pixel_disabled_struct Struct Reference

Pixels disabled in HV and/or trigger.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

  The telescope ID number (1 ... n)
- int num_trig_disabled

  Number of pixels with trigger disabled.
- int trigger_disabled [H_MAX_PIX]

  List of pixel IDs where only the trigger was disabled. We may still have signal in them.
- int num_HV_disabled

  Number of pixels with no signal (HV disabled).
- int HV_disabled [H_MAX_PIX]

  List of pixel IDs where we don't (expect to) get any signal.

6.60.1 Detailed Description

Pixels disabled in HV and/or trigger.

6.60.2 Field Documentation

6.60.2.1 HV_disabled

```
int simtel_pixel_disabled_struct::HV_disabled[H_MAX_PIX]
```

List of pixel IDs where we don't (expect to) get any signal.

No contribution towards a trigger, no matter if disabled for trigger or not. For pixels disabled afterwards, any signal should be ignored in the analysis - but they might have contributed to a trigger originally.

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

- io_hess.h

6.61 simtel_pixel_list Struct Reference

Lists of pixels (triggered, selected, etc.)

```
#include <io_hess.h>
```

Data Fields

- int code

  Indicates what sort of list this is: 0 (triggered pixel), 1 (selected pixel), ...
- int pixels

  The size of the pixels in this list.
- int pixel_list [H_MAX_PIX]

  The actual list of pixel numbers.

6.61.1 Detailed Description

Lists of pixels (triggered, selected, etc.)

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

- io_hess.h

6.62 simtel_pixel_setting_struct Struct Reference

Settings of pixel HV and thresholds.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

  The telescope ID number (1 ... n)
- int setup_id

  So far always zero.
- int trigger_mode

  So far always zero.
- int min_pixel_mult

  The minimum number of pixels in a camera.
- int num_pixels

  Local copy of the number of pixels.
- int pixel_HV_DAC [H_MAX_PIX]

  High voltage DAC values set.
- int num_drawers

  Local copy of the number of drawers (hardware modules) in the camera.
- int threshold_DAC [H_MAX_DRAWERS]

Threshold DAC values set (see detailed notes).

- int ADC_start [H_MAX_DRAWERS]

  See detailed notes for threshold_DAC.
- int ADC_count [H_MAX_DRAWERS]

  See detailed notes for threshold_DAC.
- double time_slice

  Width of readout time slice (i.e. one sample) [ns].
- int sum_bins

  Standard integration or readout or peak search over so many time slices.
- int sum_offset

  How many time slices this is supposed to start before telescope trigger.
- int nrefshape

  Number of following reference pulse shapes (num_gains or 0)
- int lrefshape

  Length of following reference pulse shape(s).
- double refshape [H_MAX_GAINS][H_MAX_FSHAPE]

  Reference pulse shape(s).
- double ref_step

  Time step between refshape entries [ns].

6.62.1 Detailed Description

Settings of pixel HV and thresholds.

6.62.2 Field Documentation

6.62.2.1 num_drawers

```
int simtel_pixel_setting_struct::num_drawers
```

Local copy of the number of drawers (hardware modules) in the camera.

Not always filled. Use the number from CameraOrganisation if you need the actual number of drawers installed in the camera. A zero here means that the following per-drawer values were not filled.

6.62.2.2 threshold_DAC

```
int simtel_pixel_setting_struct::threshold_DAC[H_MAX_DRAWERS]
```

Threshold DAC values set (see detailed notes).

This variable as well as ADC_start and ADC_count are outdated/unused, as there is no guarantee that, in a camera with different settings in different pixels, these values would be common for all pixels in the same drawer (hardware module). Usually not filled.

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

- io_hess.h

6.63 simtel_pixel_timing_struct Struct Reference

Time and amplitude values from a 'firmware'-like simple pulse analysis.

```
#include <io_hess.h>
```

Data Fields

- int known
  is pixel timing data known?
- int tel_id
  Telescope ID.
- int num_pixels
  Pixels in camera: list should be in this range.
- int num_gains
  Number of different gains per pixel.
- int list_type
  0: not set; 1: individual pixels; 2: pixel ranges.
- int list_size
  The size of the pixels in this list.
- int pixel_list [2 ∗H_MAX_PIX]
  The actual list of pixel numbers.
- int threshold
  Minimum base-to-peak raw amplitude difference applied in pixel selection.
- int before_peak
  Number of bins before peak being summed up.
- int after_peak
  Number of bins after peak being summed up.
- int num_types
  How many different types of times can we store?
- int time_type [H_MAX_PIX_TIMES]
  Which types come in which order.
- float time_level [H_MAX_PIX_TIMES]
  The width and startpos types apply.
- float granularity
  Actually stored are the following timvals divided by granularity, as 16-bit integers.
- float peak_global
  Camera-wide (mean) peak position [time slices].
- float timval [H_MAX_PIX][H_MAX_PIX_TIMES]
  Only the first 'pixels'.
- int pulse_sum_loc [H_MAX_GAINS][H_MAX_PIX]
  Amplitude sum around.
- int pulse_sum_glob [H_MAX_GAINS][H_MAX_PIX]
  Amplitude sum around.

6.63.1 Detailed Description

Time and amplitude values from a 'firmware'-like simple pulse analysis.

The structure holding these kinds timing data and the corresponding pulse sums are a MC-only add-on aimed at providing simple analysis tools with some numbers. Actual data analysis may derive similar features from the full signal traces. Don't expect any camera firmware/software to derive these numbers before the read-out.

6.63.2 Field Documentation

6.63.2.1 granularity

`float simtel_pixel_timing_struct::granularity`

Actually stored are the following timvals divided by granularity, as 16-bit integers.

Set this to e.g. 0.25 for a 0.25 time slice stepping.

6.63.2.2 pulse_sum_glob

`int simtel_pixel_timing_struct::pulse_sum_glob[H_MAX_GAINS][H_MAX_PIX]`

Amplitude sum around.

global peak; for all pixels. Ped. subtracted. Only present if before&after_peak>=0 and if list is of size>0 (otherwise no peak).

6.63.2.3 pulse_sum_loc

`int simtel_pixel_timing_struct::pulse_sum_loc[H_MAX_GAINS][H_MAX_PIX]`

Amplitude sum around.

local peak, for pixels in list. Ped. subtr. Only present if before&after_peak>=0.

6.63.2.4 time_level

`float simtel_pixel_timing_struct::time_level[H_MAX_PIX_TIMES]`

The width and startpos types apply.

above some fraction from base to peak.

Referenced by nb_fc_shaped_peak_integration(), and pixel_timing_analysis().

6.63.2.5 timval

`float simtel_pixel_timing_struct::timval[H_MAX_PIX][H_MAX_PIX_TIMES]`

Only the first 'pixels'.

elements are actually filled and stored. Others are undefined.

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

- io_hess.h

6.64 simtel_pixeltrg_time_struct Struct Reference

Times when pixels fired (not applicable for all trigger types).

```
#include <io_hess.h>
```

Data Fields

- int known

  is pixel timing data known?
- int tel_id

  Telescope ID.
- double time_step

  Time interval [ns] after telescope trigger in which times are reported.
- int num_times

  Number of fired discriminators for which time gets reported.
- int pixel_list [H_MAX_PIX]

  List of pixels IDs for which times get reported.
- int pixel_time [H_MAX_PIX]

  Time when pixel disciminator/comparator fired, in units of given time interval since telescope trigger.

6.64.1 Detailed Description

Times when pixels fired (not applicable for all trigger types).

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

- io_hess.h

6.65 simtel_pointing_correction_struct Struct Reference

Pointing correction parameters.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

  The telescope ID number (1 ... n)
- int **function_type**
- int **num_param**
- double **pointing_param** [20]

6.65.1 Detailed Description

Pointing correction parameters.

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

- io_hess.h

6.66 simtel_run_end_mc_statistics_struct Struct Reference

MC end-of-run statistics.

```
#include <io_hess.h>
```

Data Fields

- int run_num

  Run number.
- int num_showers

  Number of simulated showers found.
- int num_events

  Number of MC events found.

6.66.1 Detailed Description

MC end-of-run statistics.

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

- io_hess.h

6.67 simtel_run_end_statistics_struct Struct Reference

End-of-run statistics.

```
#include <io_hess.h>
```

Data Fields

- int run_num

    Run number.
- int num_tel

    Number of telescopes used.
- int tel_ids [H_MAX_TEL]

    IDs of all telescopes.
- int num_central_trig

    Number of system triggers.
- int num_local_trig [H_MAX_TEL]

    Number of local telescope triggers.
- int num_local_sys_trig [H_MAX_TEL]

    Number of valid telescope triggers.
- int num_events [H_MAX_TEL]

    Number of events read out.

6.67.1 Detailed Description

End-of-run statistics.

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

- io_hess.h

6.68 simtel_run_header_struct Struct Reference

Run header common to measured and simulated data.

```
#include <io_hess.h>
```

Data Fields

- int run

    Recorded data:
- time_t time

    Time of run start [UTC sec since 1970.0].
- int run_type

    Data/pedestal/laser/muon run or MC run: MC run: -1, Data run: 1, Pedestal run: 2, Laser run: 3, Muon run: 4.
- int tracking_mode

    Tracking/pointing mode: 0: Az/Alt, 1: R.A.
- int reverse_flag

    Normal or reverse tracking: 0: Normal, 1: reverse.
- double direction [2]

    Tracking/pointing direction in [radians]: [0]=Azimuth, [1]=Altitude in mode 0, [0]=R.A., [1]=Declination in mode 1.
- double offset_fov [2]

    Offset of pointing dir.
- double conv_depth

Atmospheric depth of convergence point.
- double conv_ref_pos [2]

  Reference position for convergent pointing.
- int ntel

  Number of telescopes involved.
- int tel_id [H_MAX_TEL]

  ID numbers of telescopes used in this run.
- double tel_pos [H_MAX_TEL][3]

  x,y,z positions of the telescopes [m].
- int min_tel_trig

  Minimum number of tel. in system trigger.
- int duration

  Nominal duration of run [s].
- char ∗ target

  Primary target object name.
- char ∗ observer

  Observer(s) starting or supervising run.
- int max_len_target

  For internal data handling only:
- int **max_len_observer**

6.68.1 Detailed Description

Run header common to measured and simulated data.

6.68.2 Field Documentation

6.68.2.1 conv_depth

```
double simtel_run_header_struct::conv_depth
```

Atmospheric depth of convergence point.

In [g/cm^2] from the top of the atmosphere along the system viewing direction. Typically 0 for parallel viewing or about Xmax(0.x TeV) for convergent viewing.

6.68.2.2 conv_ref_pos

```
double simtel_run_header_struct::conv_ref_pos[2]
```

Reference position for convergent pointing.

X,y in [m] at the telescope reference height.

6.68.2.3 offset_fov

```
double simtel_run_header_struct::offset_fov[2]
```

Offset of pointing dir.

in camera f.o.v. divided by focal length, i.e. converted to [radians]: [0]=Camera x (downwards in normal pointing, i.e. increasing Alt, [1]=Camera y -> Az).

6.68.2.4 run

```
int simtel_run_header_struct::run
```

Recorded data:

Run number.

Referenced by hesscam_ps_plot().

6.68.2.5 tel_pos

```
double simtel_run_header_struct::tel_pos[H_MAX_TEL][3]
```

x,y,z positions of the telescopes [m].

x is counted from array reference position towards North, y towards West, z upwards.

6.68.2.6 tracking_mode

```
int simtel_run_header_struct::tracking_mode
```

Tracking/pointing mode: 0: Az/Alt, 1: R.A.

/Dec. 2000

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

- io_hess.h

6.69 simtel_shower_parameter Struct Reference

Reconstructed shower parameters.

```
#include <io_hess.h>
```

Data Fields

- int **known**
- int num_trg

  Number of telescopes contributing to central trigger.
- int num_read

  Number of telescopes read out.
- int num_img

  Number of images used for shower parameters.
- int img_pattern

  Bit pattern of which telescopes were used (for small no. of telescopes only).
- int img_list [H_MAX_TEL]

  With more than 16 or 32 telescopes, we can only use the list.
- int result_bits

  Bit pattern of what results are available: Bits 0 + 1: direction + errors Bits 2 + 3: core position + errors Bits 4 + 5: mean scaled image shape + errors Bits 6 + 7: energy + error Bits 8 + 9: shower maximum + error
- double Az

  Azimuth angle [radians from N->E].
- double Alt

  Altitude [radians].
- double err_dir1

  Error estimate in nominal plane X direction (|| Alt) [rad].
- double err_dir2

  Error estimate in nominal plane Y direction (|| Az) [rad].
- double err_dir3

  ?
- double xc

  X core position [m].
- double yc

  Y core position [m].
- double err_core1

  Error estimate in X coordinate [m].
- double err_core2

  Error estimate in Y coordinate [m].
- double err_core3

  ?
- double mscl

  Mean scaled image length [gammas ∼1 (HEGRA-style) or ∼0 (HESS-style)].
- double **err_mscl**
- double mscw

  Mean scaled image width [gammas ∼1 (HEGRA-style) or ∼0 (HESS-style)].
- double **err_mscw**
- double energy

  Primary energy [TeV], assuming a gamma.
- double **err_energy**
- double xmax

  Atmospheric depth of shower maximum [g/cm^2].
- double **err_xmax**

6.69.1 Detailed Description

Reconstructed shower parameters.

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

- io_hess.h

6.70 simtel_tel_event_adc_struct Struct Reference

ADC data (either sampled or sum mode)

```
#include <io_hess.h>
```

Data Fields

- int known

  Must be set to 1 if and only if raw data is available.
- int tel_id

  Must match the expected telescope ID when reading.
- int num_pixels

  The number of pixels in the camera (as in configuration)
- int num_gains

  The number of different gains per pixel (2 for HESS).
- int num_samples

  The number of samples (time slices) recorded.
- int zero_sup_mode

  The desired or used zero suppression mode.
- int data_red_mode

  The desired or used data reduction mode.
- int offset_hg8

  The offset to be used in shrinking high-gain data.
- int scale_hg8

  The scale factor (denominator) in shrinking h-g data.
- int threshold

  Threshold (in high gain) for recording low-gain data.
- int list_known

  Was list of significant pixels filled in?
- int list_size

  Size of the list of available pixels (with list mode).
- int adc_list [H_MAX_PIX]

  List of available pixels (with list mode).
- uint8_t significant [H_MAX_PIX]

  Was amplitude large enough to record it? Bit 0: sum, 1: samples.
- uint8_t adc_known [H_MAX_GAINS][H_MAX_PIX]

  Was individual channel recorded? Bit 0: sum, 1: samples, 2: ADC was in saturation.
- uint32_t adc_sum [H_MAX_GAINS][H_MAX_PIX]

  Sum of ADC values.
- uint16_t adc_sample [H_MAX_GAINS][H_MAX_PIX][H_MAX_SLICES]

  Pulses sampled.

6.70.1 Detailed Description

ADC data (either sampled or sum mode)

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

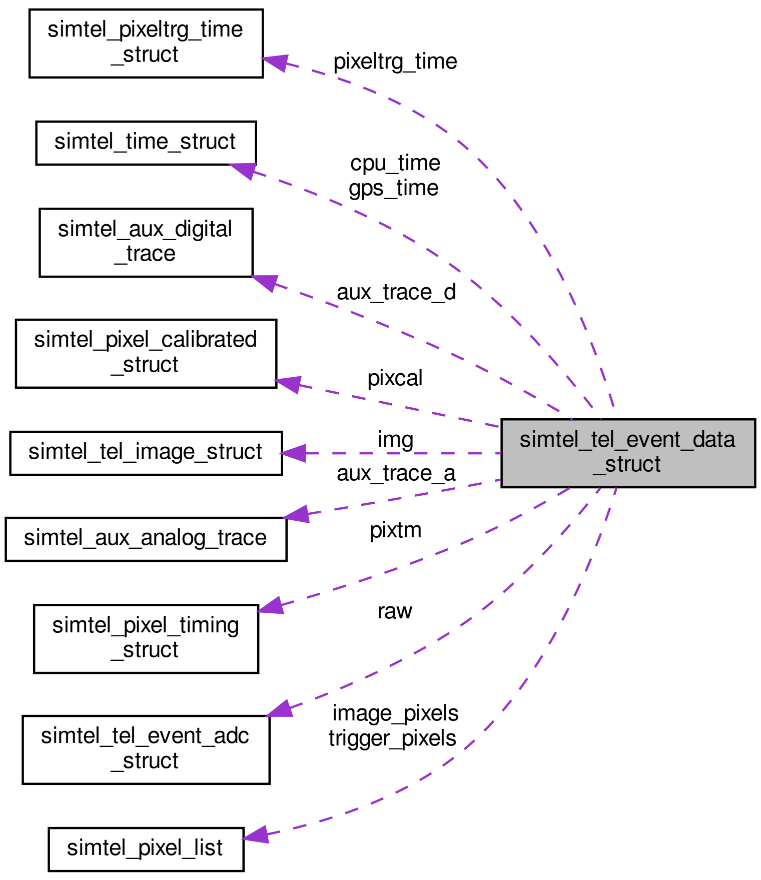
- io_hess.h

6.71 simtel_tel_event_data_struct Struct Reference

Event raw and image data from one telescope.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_tel_event_data_struct:


simtel_pixeltrg_time _struct
pixeltrg_time
simtel_time_struct
cpu_time gps_time
simtel_aux_digital _trace
aux_trace_d
simtel_pixel_calibrated _struct
pixcal
simtel_tel_image_struct
img
simtel_tel_event_data _struct
aux_trace_a
simtel_aux_analog_trace
pixtm
simtel_pixel_timing _struct
raw
simtel_tel_event_adc _struct
image_pixels trigger_pixels
simtel_pixel_list

Data Fields

- int **known**
- int tel_id

  The telescope ID number (1 ... n)
- int loc_count

  The counter for local triggers.
- int glob_count

  The counter for system triggers.
- HTime cpu_time

  Camera CPU system time of event.
- HTime gps_time

  GPS time of event, if any.
- int start_readout

  Position in simulated memory where readout starts.
- double time_readout

  Time when readout starts (+array-wide arbitrary offset) [ns].
- double time_trg_rel

  Time of telescope trigger relative to start of readout. [ns].
- int trg_source

  1=internal (event data) or 2=external (calib data).
- int num_list_trgsect

  Number of trigger groups (sectors) listed.
- int list_trgsect [H_MAX_SECTORS]

  List of triggered groups (sectors).
- int known_time_trgsect

  Are the trigger times known? (0/1)
- double time_trgsect [H_MAX_SECTORS]

  Times when trigger groups (as in list) fired.
- int readout_mode

  Sum mode (0) or sample mode only (1) or both (>=2)
- int num_image_sets

  how many 'img' sets are available.
- int max_image_sets

  how many 'img' sets were allocated.
- AdcData ∗ raw

  Pointer to raw data, if any.
- PixelTiming ∗ pixtm

  Optional pixel (pulse shape) timing.
- ImgData ∗ img

  Pointer to second moments, if any.
- PixelCalibrated ∗ pixcal

  Pointer to calibrated pixel intensities, if available.
- int num_phys_addr

  (not used)
- int phys_addr [4 ∗H_MAX_DRAWERS]

  (not used)
- PixelList trigger_pixels

  List of triggered pixels.
- PixelList image_pixels

  Pixels included in (first) image.

- PixelTrgTime pixeltrg_time

  Times when individual pixels fired.
- AuxTraceD aux_trace_d [MAX_AUX_TRACE_D]

  Optional auxiliary digital traces.
- AuxTraceA aux_trace_a [MAX_AUX_TRACE_A]

  Optional auxiliary analog traces.

6.71.1 Detailed Description

Event raw and image data from one telescope.

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

- io_hess.h

6.72 simtel_tel_image_struct Struct Reference

Image parameters.

```
#include <io_hess.h>
```

Data Fields

- int known

  is image data known?
- int tel_id

  Telescope ID.
- int pixels

  number of pixels used for image
- int cut_id

  For which set of tail-cuts was used.
- double amplitude

  Image amplitude (="SIZE") [mean p.e.].
- double clip_amp

  Pixel amplitude clipping level [mean p.e.] or zero for no clipping.
- int num_sat

  Number of pixels in saturation (ADC saturation or dedicated clipping).
- double x

  Position.
- double x_err

  Error on x (0: error not known, <0: x not known) [rad].
- double y

  Y position (c.o.g.) [rad], corrected for any camera rotation.
- double y_err

  Error on y (0: error not known, <0: y not known) [rad].
- double phi

  Orientation.

- double phi_err

  Error on phi (0: error not known, <0: phi not known) [rad].
- double l

  Shape.
- double l_err

  Error on length (0: error not known, <0: l not known) [rad].
- double w

  Width (minor axis) [rad].
- double w_err

  Error on width (0: error not known, <0: w not known) [rad].
- double skewness

  Skewness, indicating asymmetry of image.
- double skewness_err

  Error (0: error not known, <0: skewness not known)
- double kurtosis

  Kurtosis, indicating sharpness of peak of image.
- double kurtosis_err

  Error (0: error not known, <0: kurtosis not known)
- int num_conc

  Number of hottest pixels used for concentration.
- double concentration

  Fraction of total amplitude in num_conc hottest pixels.
- double tm_slope

  Timing.
- double tm_residual

  R.m.s. average residual time after slope correction. [ns].
- double tm_width1

  Average pulse width (50% of peak or time over threshold) [ns].
- double tm_width2

  Average pulse width (20% of peak or 0) [ns].
- double tm_rise

  Average pixel rise time (or 0) [ns].
- int num_hot

  Individual pixels.
- int hot_pixel [H_MAX_HOTPIX]

  Pixel IDs of hotest pixels.
- double hot_amp [H_MAX_HOTPIX]

  Amplitudes of hotest pixels [mean p.e.].

6.72.1 Detailed Description

Image parameters.

6.72.2 Field Documentation

6.72.2.1 l

```
double simtel_tel_image_struct::l
```

Shape.

Length (major axis) [rad]

6.72.2.2 num_hot

```
int simtel_tel_image_struct::num_hot
```

Individual pixels.

Number of hottest pixels individually saved

6.72.2.3 phi

```
double simtel_tel_image_struct::phi
```

Orientation.

Angle of major axis w.r.t. x axis [rad], corrected for any camera rotation.

6.72.2.4 tm_slope

```
double simtel_tel_image_struct::tm_slope
```

Timing.

Slope in peak times along major axis as given by phi. [ns/rad]

Referenced by pixel_timing_analysis().

6.72.2.5 x

```
double simtel_tel_image_struct::x
```

Position.

X position (c.o.g.) [rad], corrected for any camera rotation.

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

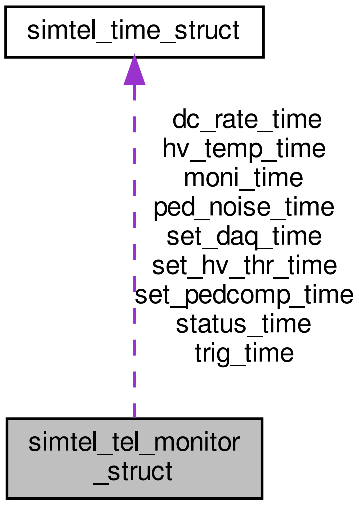
- io_hess.h

6.73 simtel_tel_monitor_struct Struct Reference

Monitoring data, traditionally emulating first-generation HESS cameras.

```
#include <io_hess.h>
```

Collaboration diagram for simtel_tel_monitor_struct:



Data Fields

- int known

  Status etc., pedestals, DC, HV.
- int new_parts

  What of that is new.
- int tel_id

  Telescope ID number.
- int num_sectors

  Number of sector available for trigger (default trigger).
- int num_pixels

  Number of pixels in camera.
- int num_drawers

  Number of drawers in camera.
- int num_gains

  Number of different electronics gains for read-out (1 or 2).
- int num_ped_slices

  How many slices have been added for pedestal.
- int num_drawer_temp

  Number of temperatures per drawer.
- int num_camera_temp

Number of other temperatures monitored.

- int monitor_id

  Incremented with each update.
- HTime moni_time

  Time when last monitoring data was sent.
- HTime **status_time**
- HTime trig_time

  Time when last trigger monitor data was read.
- HTime ped_noise_time

  Time when pedestals + noise were determined.
- HTime hv_temp_time

  Time when hv+currents+temp. were all read out.
- HTime dc_rate_time

  Time when DC current + pixels scalers were read.
- HTime set_hv_thr_time

  Time when HV + thresholds where set.
- HTime set_daq_time

  Time when DAQ parameters where set.
- HTime set_pedcomp_time

  Time when pedestal compensations where set.
- int status_bits

  Lid, HV, trigger, readout, drawers, fans.
- long coinc_count

  These have to be obtained from the camera trigger electronics (first trigger type only)
- long event_count

  Count of events read out.
- double event_rate

  Average event rate [Hz].
- double data_rate

  Average rate of packed data [MB/s].
- double trigger_rate

  Camera average local trigger rate [Hz].
- double sector_rate [H_MAX_SECTORS]

  Sector trigger rate [Hz].
- double mean_significant

  These are computed by the readout software:
- double pedestal [H_MAX_GAINS][H_MAX_PIX]

  Average pedestal on ADC sums.
- double pedsamp [H_MAX_GAINS][H_MAX_PIX]

  Corresponding pedestal per sample.
- double noise [H_MAX_GAINS][H_MAX_PIX]

  Average noise on ADC sums.
- int ped_comp_rel [H_MAX_GAINS][H_MAX_PIX]

  Pedestal compensation (optional)
- uint16_t current [H_MAX_PIX]

  These numbers need mapping from drawers+channel to pixel id:
- uint16_t scaler [H_MAX_PIX]

  ADC values of pixel trigger rate.
- uint16_t hv_v_mon [H_MAX_PIX]

  ADC values of HV voltage monitor.

- uint16_t hv_i_mon [H_MAX_PIX]

  ADC values of HV current monitor.
- uint16_t hv_dac [H_MAX_PIX]

  DAC values of HV settings.
- uint16_t thresh_dac [H_MAX_DRAWERS]

  Thresholds set in each drawer.
- uint8_t trig_set [H_MAX_PIX]

  Set if pixel excluded from trigger.
- uint8_t hv_set [H_MAX_PIX]

  Set if HV switched off for pixel.
- uint8_t hv_stat [H_MAX_PIX]

  Set if HV switched off for pixel.
- short drawer_temp [H_MAX_DRAWERS][H_MAX_D_TEMP]

  That is left in its raw order:
- short camera_temp [H_MAX_C_TEMP]

  ADC values.
- uint16_t daq_conf

  As set by CNTRLDaq message.
- uint16_t **daq_scaler_win**
- uint16_t **daq_nd**
- uint16_t **daq_acc**
- uint16_t **daq_nl**

6.73.1 Detailed Description

Monitoring data, traditionally emulating first-generation HESS cameras.

6.73.2 Field Documentation

6.73.2.1 coinc_count

```
long simtel_tel_monitor_struct::coinc_count
```

These have to be obtained from the camera trigger electronics (first trigger type only)

Count of pixel coincidences (local triggers).

6.73.2.2 current

```
uint16_t simtel_tel_monitor_struct::current[H_MAX_PIX]
```

These numbers need mapping from drawers+channel to pixel id:

ADC values of DC current.

6.73.2.3 drawer_temp

```
short simtel_tel_monitor_struct::drawer_temp[H_MAX_DRAWERS][H_MAX_D_TEMP]
```

That is left in its raw order:

ADC values.

6.73.2.4 known

```
int simtel_tel_monitor_struct::known
```

Status etc., pedestals, DC, HV.

That includes:

- 0x01 (Status only)
- 0x02 (Counts + Rates)
- 0x04 (Pedestals + noise)
- 0x08 (HV + temperatures)
- 0x10 (Pixel scalers + DC currents)
- 0x20 (HV + thresholds settings)
- 0x40 (DAQ configuration)
- 0x80 (Pedestal compensation)

Referenced by write_simtel_tel_monitor().

6.73.2.5 ped_comp_rel

```
int simtel_tel_monitor_struct::ped_comp_rel[H_MAX_GAINS][H_MAX_PIX]
```

Pedestal compensation (optional)

Values added to ADC counts for pedestal compensation

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

- io_hess.h

6.74 simtel_time_struct Struct Reference

Breakdown of time into seconds since 1970.0 and nanoseconds.

```
#include <io_hess.h>
```

Data Fields

- long **seconds**
- long **nanoseconds**

6.74.1 Detailed Description

Breakdown of time into seconds since 1970.0 and nanoseconds.

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

- io_hess.h

6.75 simtel_tracking_event_data_struct Struct Reference

Tracking data interpolated for one event and one telescope.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

  The telescope ID number (1 ... n)
- double azimuth_raw

  Raw azimuth angle [radians from N->E].
- double altitude_raw

  Raw altitude angle [radians].
- double azimuth_cor

  Azimuth corrected for pointing errors.
- double altitude_cor

  Azimuth corrected for pointing errors.
- int raw_known

  Set if raw angles are known.
- int cor_known

  Set if corrected angles are known.

6.75.1 Detailed Description

Tracking data interpolated for one event and one telescope.

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

- io_hess.h

6.76 simtel_tracking_setup_struct Struct Reference

Definition of tracking parameters.

```
#include <io_hess.h>
```

Data Fields

- int tel_id

  Telescope ID.
- int **known**
- int drive_type_az

  0 for now.
- int drive_type_alt

  0 for now.
- double zeropoint_az

  Offsets subtracted from the values reported.
- double zeropoint_alt

  by hardware before calculating 'raw' angles [rad].
- double sign_az

  This is -1 if hardware counts the other way than.
- double sign_alt

  we do, and +1 otherwise.
- double resolution_az

  Typical resolution expected [rad].
- double resolution_alt

  Typical resolution expected [rad].
- double range_low_az

  Note: The values may be outside the [0...2∗pi[range.
- double **range_low_alt**
- double **range_high_az**
- double **range_high_alt**
- double **park_pos_az**
- double **park_pos_alt**

6.76.1 Detailed Description

Definition of tracking parameters.

This is a copy of the configuration given to the tracking computers. Note: all angles are in radians. This block should not be needed for event analysis.

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

- io_hess.h

6.77 tel_type_param Struct Reference

Data Fields

- int **min_tel_id**
- int **max_tel_id**
- double **mirror_area**
- double **flen**
- int **num_pixels**

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

- user_analysis.c

6.78 telescope_list Struct Reference

Data Fields

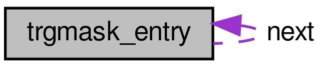
- size_t **min_tel**
- size_t **ntel**
- int $*$ **tel_id**

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

- user_analysis.c

6.79 trgmask_entry Struct Reference

Collaboration diagram for trgmask_entry:



Data Fields

- long event

  The event number.
- int tel_id

  The telescope ID number.
- int trg_mask

  The trigger mask bit pattern which got messed up in data files.
- struct trgmask_entry ∗ next

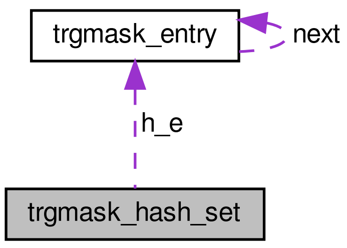
  Can be used in arrays but also in linked lists.

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

- io_trgmask.h

6.80 trgmask_hash_set Struct Reference

Collaboration diagram for trgmask_hash_set:



Data Fields

- long **run**
- struct trgmask_entry ∗ h_e [TRGMASK_PRIME]

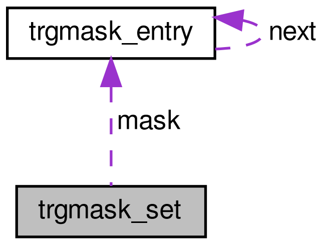
  Start of linked list for each possible hash value.

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

- io_trgmask.h

6.81 trgmask_set Struct Reference

Collaboration diagram for trgmask_set:



Data Fields

- long **run**
- size_t **num_entries**
- struct trgmask_entry ∗ **mask**

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

- io_trgmask.h

6.82 user_parameters Struct Reference

Data Fields

-

  struct {
    int user_flags
      1: HESS-style analysis standard cuts; 2: hard cuts; 3: loose cuts.
    int min_pix
      The minimum number of significant pixels in usable images.
    int reco_flag
      Reconstruction level flag.
    int min_tel_img
      Minimum and maximum number of usable images for events used in analysis.
    int **max_tel_img**
    int lref
      Which pixel's amplitude is used as reference.
    int integrator
      The type of pixel intensity integration scheme.
    int integ_param [3]
      Integration-scheme-specific integer parameters, typically:
    int integ_thresh [2]

Integer type thresholds for significance in ADC units (one per gain)
int integ_no_rescale
Set to 1 if integration over small window should not rescale for fraction of single p.e.
int trg_req
Required trigger type (bit pattern: bit 0 = majo, 1=asum, 2=dsum, 3=dmajo)
int pixstat
Evaluate pixel and/or trigger group trigger efficiency statistics.
} **i**

•

struct {
double **source_offset_deg**
double d_sp_idx
Difference between generated MC spectrum (e.g.
double min_amp
The minimum amplitude [peak p.e.
double tailcut_low
The lower and upper tail cuts for the standard two-level tail-cut scheme.
double **tailcut_high**
double minfrac
Minimum fraction of reference amplitude is needed.
double **max_theta_deg**
double **theta_scale**
double **de2_cut_param** [4]
double **mscrw_min** [4]
double **mscrw_max** [4]
double **mscrl_min** [4]
double **mscrl_max** [4]
double **eres_cut_param** [4]
double **hmax_cut_param**
double **min_theta_deg**
double camera_clipping_deg
Pixel outside this radius (if $>$ 0) should be ignored in image reconstruction.
double theta_escale [4]
If the angular acceptance deviates from the 80% containment.
double clip_amp
Pixel intensity clipped to this value after calibration, if this param is not zero.
double d_integ_param [2][4]
Integration-scheme- and gain-specific floating-point parameters.
double calib_scale
Calibration scale from mean-p.e.
double r_nb [3]
Radii for initial neighbour pixel search.
double r_ne
Radius for extending significant pixels in image cleaning [pixel diameter].
double impact_range [3]
[0]: maximum distance of array center from shower axis, [1],[2]: max.
double true_impact_range [3]
As for impact_ranhe.
double **max_core_distance**
double **focal_length**
} **d**

6.82.1 Field Documentation

6.82.1.1 calib_scale

```
double user_parameters::calib_scale
```

Calibration scale from mean-p.e.

units to experimental units (0.0: like HESS).

6.82.1.2 d_sp_idx

```
double user_parameters::d_sp_idx
```

Difference between generated MC spectrum (e.g.

E^-2.0) and assumed source spectrum (e.g. E^-2.5), e.g. case d_sp_idx = -0.5 .

6.82.1.3 impact_range

```
double user_parameters::impact_range[3]
```

[0]: maximum distance of array center from shower axis, [1],[2]: max.

|x|,|y| of core in ground plane.

6.82.1.4 integ_no_rescale

```
int user_parameters::integ_no_rescale
```

Set to 1 if integration over small window should not rescale for fraction of single p.e.

trace.

6.82.1.5 integ_param

```
int user_parameters::integ_param[3]
```

Integration-scheme-specific integer parameters, typically:

number of bins to integrate and some offset value from start or back from detected peak.

6.82.1.6 integrator

```
int user_parameters::integrator
```

The type of pixel intensity integration scheme.

0: none (implicitly all samples), 1: simple, 2: around global peak, 3: around local peak, 4: around peak in neighbour pixels.

6.82.1.7 min_amp

```
double user_parameters::min_amp
```

The minimum amplitude [peak p.e.

] of images usable for the analysis.

6.82.1.8 r_nb

```
double user_parameters::r_nb[3]
```

Radii for initial neighbour pixel search.

Maximum search radii for neighbours [pixel diameter]

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

- user_analysis.h

6.83 warn_specific_data Struct Reference

A struct used to store thread-specific data.

Data Fields

- int **warninglevel**
- int **warningmode**
- char **output_buffer** [2048]
- const char ∗ logfname

  The name of the log file.
- char **saved_logfname** [256]
- int **buffered**
- FILE ∗ **logfile**
- void(∗ **log_function**)(const char ∗, const char ∗, int, int)
- void(∗ **output_function**)(const char ∗)
- char ∗(∗ **aux_function**)(void)
- int **recursive**

6.83.1 Detailed Description

A struct used to store thread-specific data.

6.83.2 Field Documentation

6.83.2.1 logfname

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

The name of the log file.

Used only when opening the file.

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

- warning.c

Chapter 7

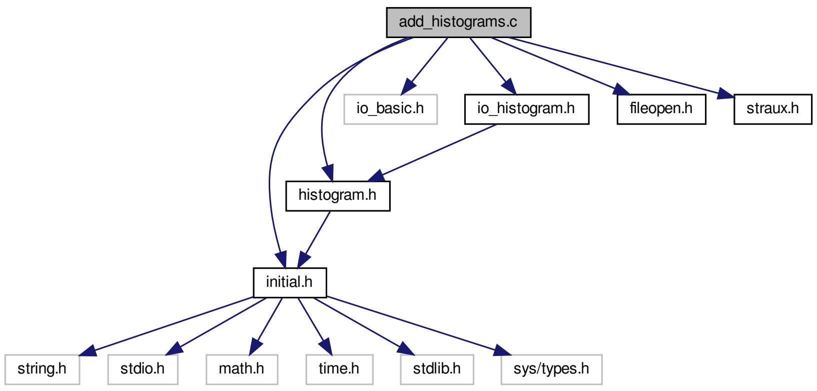
File Documentation

7.1 add_histograms.c File Reference

Utility program for adding up matching histograms.

```
#include "initial.h"
#include "histogram.h"
#include "io_basic.h"
#include "io_histogram.h"
#include "fileopen.h"
#include "straux.h"
```

Include dependency graph for add_histograms.c:



Functions

- void **syntax** (const char ∗prgm)
- int main (int argc, char ∗∗argv)

    Main program.

7.1.1 Detailed Description

Utility program for adding up matching histograms.

```
Utility program for adding up matching histograms.

Syntax:  add_histograms [ -x id1,...] input_files ... -o output_file
```

The histograms may be within multiple I/O blocks of the input file. Matching histograms will be added up, unless set to be excluded with the '-x' option. Only non-empty histograms are written to output.

Author

Konrad Bernloehr

Date

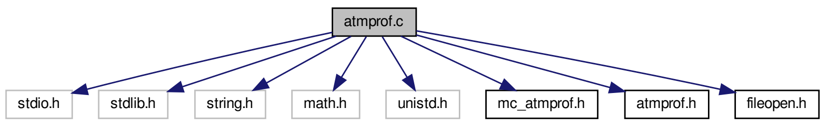
2013 to 2022

7.2 atmprof.c File Reference

A stripped-down version of the interpolation of atmospheric profiles from the atmo.c file of the CORSIKA IACT/ATMO package.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include "mc_atmprof.h"
#include "atmprof.h"
#include "fileopen.h"
```

Include dependency graph for atmprof.c:



Macros

- #define **MAX_PROFILE** 50

Functions

- static void interp (double x, double ∗v, int n, int ∗ipl, double ∗rpl)

  Linear interpolation with binary search algorithm.
- static double rpol (double ∗x, double ∗y, int n, double xp)

  Linear interpolation with binary search algorithm.
- static char ∗ find_elsewhere (const char ∗fname, char ∗bf, size_t sz)

  Find the atmospheric profiles elsewhere (in the sim_telarray configuration).
- int init_atmprof (int atmosphere)

  Initialize atmospheric profiles.
- int init_atmprof_s (AtmProf ∗aprof)

  Initialize atmospheric profiles.
- 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 refidx (double height)

  Index of refraction as a function of altitude [cm].
- double heighx (double thick)

  Altitude [m] as a function of atmospheric thickness [g/cm∗∗2].

Variables

- static int **current_atmosphere**
- static int **num_prof**
- static double **p_alt** [MAX_PROFILE]
- static double **p_log_rho** [MAX_PROFILE]
- static double **p_rho** [MAX_PROFILE]
- static double **p_log_thick** [MAX_PROFILE]
- static double **p_log_n1** [MAX_PROFILE]
- static double **top_of_atmosphere** = 112.83e3
- static double **bottom_of_atmosphere** = 0.

7.2.1 Detailed Description

A stripped-down version of the interpolation of atmospheric profiles from the atmo.c file of the CORSIKA IACT/ATMO package.

The main differences are: a) parameters are passed by value instead of FORTRAN by-reference way, b) the height is measured in meters, c) interpolation is linear in log(density) etc. rather than using cubic splines like available with the repolator code.

The profiles can be set up by atmosphere number <n> (which means searching for file atmprof<n>.dat in some known paths), or from a AtmProf structure (table part or CORSIKA 5-layer parameters as fall-back). The CORSIKA built-in profiles are expanded internally to tables first, with the relevant parameters only available from the AtmProf structure.

Author

Konrad Bernloehr

Date

1990 to 2019

7.2.2 Function Documentation

7.2.2.1 heighx()

```
double heighx (
             double thick )
```

Altitude [m] as a function of atmospheric thickness [g/cm∗∗2].

Parameters

thick	atmospheric thickness [g/cm∗∗2]

Returns

altitude [m]

7.2.2.2 init_atmprof()

```
int init_atmprof (
             int atmosphere )
```

Initialize atmospheric profiles.

Atmospheric models are read in from text-format tables. For the interpolation of relevant parameters (density, thickness, index of refraction, ...) all parameters are transformed such that linear interpolation can be easily used.

Parameters

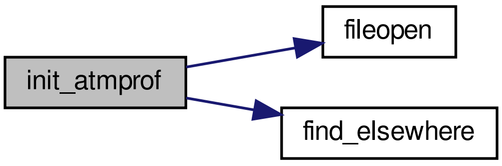
atmosphere	Atmosphere number, to be expanded to the table file name.

Returns

0 (OK) or -1 (error, e.g. table available)

References fileopen(), and find_elsewhere().

Here is the call graph for this function:



7.2.2.3 init_atmprof_s()

```
int init_atmprof_s (
            AtmProf * aprof )
```

Initialize atmospheric profiles.

Atmospheric models are passed on from the data itself. For the interpolation of relevant parameters (density, thickness, index of refraction, ...) all parameters are transformed such that linear interpolation can be easily used.

Parameters

aprof	Pointer to an AtmProf structure (can be NULL to be replaced by the common one).

Returns

0 (OK) or -1 (error, e.g. table available)

7.2.2.4 interp()

```
static void interp (
            double x,
            double * v,
            int n,
            int * ipl,
            double * rpl ) [static]
```

Linear interpolation with binary search algorithm.

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) 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

x	Input: the requested coordinate
v	Input: tabulated coordinates at data points
n	Input: number of data points
ipl	Output: the number of the data point following the requested coordinate in the given sorting (1 <= ipl <= n-1)
rpl	Output: the fraction (x-v[ipl-1])/(v[ipl]-v[ipl-1]) with 0 <= rpl <= 1

Referenced by rpol().

7.2.2.5 refidx()

```
double refidx (
             double height )
```

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

Parameters

height	altitude [m]

Returns

index of refraction

7.2.2.6 rhofx()

```
double rhofx (
             double height )
```

Density of the atmosphere as a function of altitude.

Parameters

height	altitude [m]

Returns

density [g/cm∗∗3]

7.2.2.7 rpol()

```
static double rpol (
             double * x,
             double * y,
             int n,
             double xp )  [static]
```

Linear interpolation with binary search algorithm.

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 function calls interp() to find out where to interpolate.

Parameters

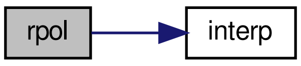
x	Input: Coordinates for data table
y	Input: Corresponding values for data table
n	Input: Number of data points
xp	Input: Coordinate of requested value

Returns

Interpolated value

References interp().

Here is the call graph for this function:



7.2.2.8 thickx()

```
double thickx (
             double height )
```

Atmospheric thickness [g/cm∗∗2] as a function of altitude.

Parameters

height	altitude [m]

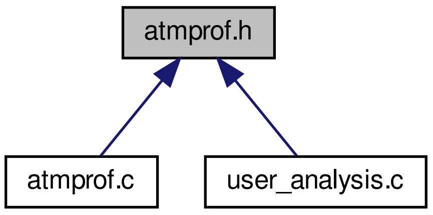
Returns

thickness [g/cm∗∗2]

7.3 atmprof.h File Reference

Function prototypes for atmprof.c.

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



Functions

- int init_atmprof (int atmosphere)

  Initialize atmospheric profiles.
- int init_atmprof_s (AtmProf ∗aprof)

  Initialize atmospheric profiles.
- 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 refidx (double height)

  Index of refraction as a function of altitude [cm].
- double heighx (double thick)

  Altitude [m] as a function of atmospheric thickness [g/cm∗∗2].

7.3.1 Detailed Description

Function prototypes for atmprof.c.

Author

Konrad Bernloehr

Date

2008 to 2019

7.3.2 Function Documentation

7.3.2.1 heighx()

```
double heighx (
            double thick )
```

Altitude [m] as a function of atmospheric thickness [g/cm∗∗2].

Parameters

thick	atmospheric thickness [g/cm∗∗2]

Returns

altitude [m]

7.3.2.2 init_atmprof()

```
int init_atmprof (
            int atmosphere )
```

Initialize atmospheric profiles.

Atmospheric models are read in from text-format tables. For the interpolation of relevant parameters (density, thickness, index of refraction, ...) all parameters are transformed such that linear interpolation can be easily used.

Parameters

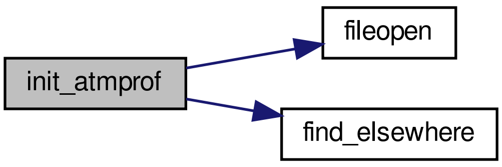
atmosphere	Atmosphere number, to be expanded to the table file name.

Returns

0 (OK) or -1 (error, e.g. table available)

References fileopen(), and find_elsewhere().

Here is the call graph for this function:



7.3.2.3 init_atmprof_s()

```
int init_atmprof_s (
            AtmProf * aprof )
```

Initialize atmospheric profiles.

Atmospheric models are passed on from the data itself. For the interpolation of relevant parameters (density, thickness, index of refraction, ...) all parameters are transformed such that linear interpolation can be easily used.

Parameters

aprof	Pointer to an AtmProf structure (can be NULL to be replaced by the common one).

Returns

0 (OK) or -1 (error, e.g. table available)

7.3.2.4 refidx()

```
double refidx (
            double height )
```

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

Parameters

height	altitude [m]

Returns

index of refraction

7.3.2.5 rhofx()

```
double rhofx (
             double height )
```

Density of the atmosphere as a function of altitude.

Parameters

height	altitude [m]

Returns

density [g/cm∗∗3]

7.3.2.6 thickx()

```
double thickx (
             double height )
```

Atmospheric thickness [g/cm∗∗2] as a function of altitude.

Parameters

height	altitude [m]

Returns

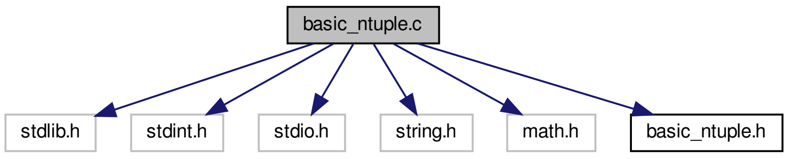
thickness [g/cm∗∗2]

7.4 basic_ntuple.c File Reference

Print specific ntuple data from read_hess shower analysis.

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "basic_ntuple.h"
```

Include dependency graph for basic_ntuple.c:



Macros

- #define **M_PI** 3.14159265358979323846

Functions

- int list_ntuple (FILE ∗f, const struct basic_ntuple ∗b, int wtr)

  List the parameters useful for event selection plus some more parameters which should not be used for event selection.

Variables

- static int **list_init** = 0
- static int **with_true** = 0

7.4.1 Detailed Description

Print specific ntuple data from read_hess shower analysis.

Author

Konrad Bernloehr

Date

2009 to 2023

7.4.2 Function Documentation

7.4.2.1 list_ntuple()

```
int list_ntuple (
            FILE * f,
            const struct basic_ntuple * b,
            int wtr )
```

List the parameters useful for event selection plus some more parameters which should not be used for event selection.

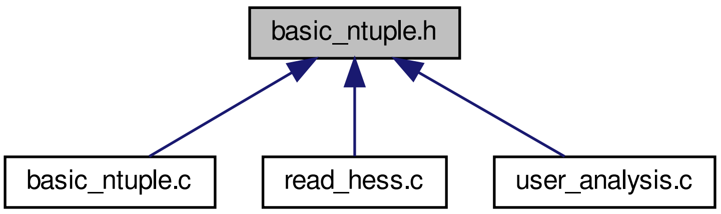
Parameters

f	Output file, to be opened beforehand.
b	Pointer to the struct containing all the relevant numbers.
wtr	Non-zero on first call to write also true MC parameters.

7.5 basic_ntuple.h File Reference

Declaration of the basic_ntuple struct.

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



Data Structures

- struct basic_ntuple

  A struct with basic per-shower parameters, to be used as an n-tuple in the event selection.

Functions

- int list_ntuple (FILE ∗f, const struct basic_ntuple ∗b, int wtr)

  List the parameters useful for event selection plus some more parameters which should not be used for event selection.

7.5.1 Detailed Description

Declaration of the basic_ntuple struct.

Date

2009, 2010

7.5.2 Function Documentation

7.5.2.1 list_ntuple()

```
int list_ntuple (
            FILE * f,
            const struct basic_ntuple * b,
            int wtr )
```

List the parameters useful for event selection plus some more parameters which should not be used for event selection.

Parameters

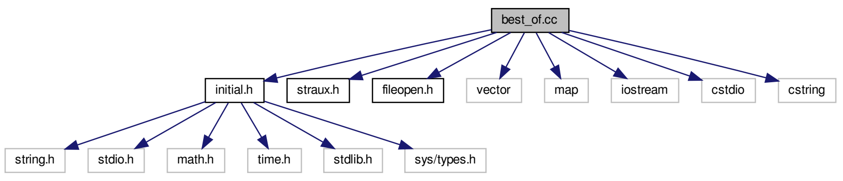
f	Output file, to be opened beforehand.
b	Pointer to the struct containing all the relevant numbers.
wtr	Non-zero on first call to write also true MC parameters.

7.6 best_of.cc File Reference

Tool for extracting best values from listings of 'rh3' sensitivity evaluations.

```
#include "initial.h"
#include "straux.h"
#include "fileopen.h"
#include <vector>
#include <map>
#include <iostream>
#include <cstdio>
#include <cstring>
```

Include dependency graph for best_of.cc:



Data Structures

- struct best_value

Enumerations

- enum **SpecType** {
  SPEC_NONE = -1 , **SPEC_GAMMA** = 0 , **SPEC_ELECTRON** = 1 , **SPEC_PROTON** = 101 ,
  SPEC_HE = 402 , **SPEC_CNO** = 1407 , **SPEC_SI** = 2814 , **SPEC_IRON** = 5626 }
- enum **espec_t** { **OLD_E_POWERLAW** = 1 , **NEW_E_POWERLAW** = 2 , **NEW_E_PL_LGN1** = 3 , **NEW_↩
  E_PL_LGN2** = 4 }
- enum **BestChoice** {
  BestDiff =1 , **BestIntegral** =2 , **BestAngle** =3 , **BestEres** =4 ,
  BestRate =5 , **BestCombined** =6 , **BestAll** =7 }

Functions

- string **particle_type** (SpecType sp)
- double **Crab_Unit** (double E)
- static double **cu** (double x)
- double **Crab_Unit_int** (double E)
- double **ergs** (double E)
- static double **f50** (double x)
- static double **fsp50** (double x)
- double **Flux_req50_south** (double E)
- double **Flux_req50_E2erg_south** (double E)
- double **Flux_req50_CU_south** (double E)
- static double **fn50** (double x)
- static double **fnsp50** (double x)
- double **Flux_req50_north** (double E)
- double **Flux_req50_E2erg_north** (double E)
- double **Flux_req50_CU_north** (double E)
- static double **f5** (double x)
- static double **fsp5** (double x)
- double **Flux_req5_south** (double E)
- double **Flux_req5_E2erg_south** (double E)
- double **Flux_req5_CU_south** (double E)
- static double **fn5** (double x)
- static double **fnsp5** (double x)
- double **Flux_req5_north** (double E)
- double **Flux_req5_E2erg_north** (double E)
- double **Flux_req5_CU_north** (double E)
- static double **f05** (double x)
- static double **fsp05** (double x)
- double **Flux_req05_south** (double E)
- double **Flux_req05_E2erg_south** (double E)
- double **Flux_req05_CU_south** (double E)
- static double **fn05** (double x)
- static double **fnsp05** (double x)
- double **Flux_req05_north** (double E)
- double **Flux_req05_E2erg_north** (double E)
- double **Flux_req05_CU_north** (double E)
- static double **fd50** (double x)

- static double **fdes50** (double x)
- double **Flux_goal50_south** (double E)
- double **Flux_goal50_E2erg_south** (double E)
- double **Flux_goal50_CU_south** (double E)
- static double **fnd50** (double x)
- static double **fndes50** (double x)
- double **Flux_goal50_north** (double E)
- double **Flux_goal50_E2erg_north** (double E)
- double **Flux_goal50_CU_north** (double E)
- double **Angular_resolution_req** (double E)
- double **Angular_resolution_goal** (double E)
- static double **eresb** (double E)
- double **Energy_resolution_req** (double E)
- static double **eresdb** (double E)
- double **Energy_resolution_goal** (double E)
- double **flux_int** (SpecType sp, double E1, double E2)
- double **lima17** (double on, double off, double alpha)
- bool **matching_required_diffsens** (int calc_pput, bool with_flux, double E, double diff_sens)
- bool **matching_required_performance** (int calc_pput, bool with_flux, double E, double diff_sens, double angres, double eres)
- bool **matching_required_angres** (double E, double angres)
- bool **matching_required_eres** (double E, double eres)
- int **main** (int argc, char **argv)

Variables

- static double **sce** = 1.6022
- static double **sca** = 1e-4
- static double **sc** = sce*sca
- espec_t **espec_type** = OLD_E_POWERLAW

7.6.1 Detailed Description

Tool for extracting best values from listings of 'rh3' sensitivity evaluations.

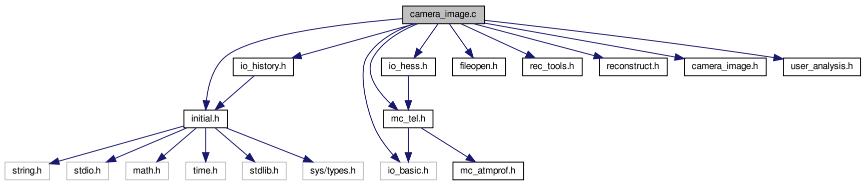
Three versions of the 'rh3' output format are supported. All of the input (from standard input) should be in the same format type.

7.7 camera_image.c File Reference

Plot a camera image from H.E.S.S.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_history.h"
#include "io_hess.h"
#include "fileopen.h"
#include "rec_tools.h"
#include "reconstruct.h"
#include "camera_image.h"
```

```
#include "user_analysis.h"
```
Include dependency graph for camera_image.c:



Data Structures

- struct primary_id_struct

Macros

- #define **NUM_LANG** 2
- #define **H_MAX_NB1** 8
- #define **H_MAX_NB2** 24

Typedefs

- typedef struct primary_id_struct **PrimaryId**

Functions

- static int **guessed_pixel_shape_type** (CameraSettings ∗camset, int itel)
- static double **dist2** (double x, double y)
- static void print_pix_col (double n_o_r, FILE ∗psfile, double gamma_coeff, int mode)

  Print a false-colour RGB value for a pixel intensity.
- static void camimg_ps_initconst ()

  Set some (to remain constant) values based on environment values.
- static void camimg_ps_header (FILE ∗psfile, const char ∗image_fname)

  Write one-time header material at the start of a new Postscript file.
- static FILE ∗ camimg_ps_open (const char ∗image_fname)

  Open the Postscript output file for camera plots.
- static int camimg_ps_pixel_def (FILE ∗psfile, CameraSettings ∗camset, int itel, double scale, double body←↩ _diameter)

  Define Postscript macros for showing a pixel of the given shape.
- static int **camimg_ps_page_header** (FILE ∗psfile, int event, CameraSettings ∗camset, int itel, double scale, double body_diameter)
- static const char ∗ **find_primary_name** (int primary_id)
- void hesscam_ps_plot (const char ∗image_fname, AllHessData ∗hsdata, int itel, int type, int amp_tm, double clip_amp)

  Write PostScript of camera sum image or sample image to a dedicated file.
- void **hesscam_type_sum_plot** (const char ∗image_fname, AllHessData ∗hsdata, int teltype)
- static int find_neighbours (CameraSettings ∗camset, int itel)

  Find the list of neighbours for each pixel.

Variables

- static char **ps_head1a** [ ]
- static char **ps_head1b** [ ]
- static char **ps_head2** [ ]
- static char **ps_head3** [ ]
- static char **ps_begin_page1** [ ]
- static char **ps_begin_page2** [ ]
- static char **ps_end_page** [ ]
- static char **ps_trailer** [ ]
- static char **alt_az_arrow** [ ]
- static int **ps_num_page** = 0
- static double **gamma_coeff** = 0.65
- static double **img_gamma** = 0.
- static double **img_range** = 20.
- static double **img_off** = 4.
- static int **with_id** = 0
- static int **with_amp** = 0
- static int **with_npe** = 0
- static int **with_sum_only** = 0
- static int **without_reco** = 0
- static int **with_show_true_pe** = 0
- static int **with_show_npe** = 0
- static int **without_pix_cross** = 0
- static char ∗ **with_plot_title** = NULL
- static const double **hex_dx** [6] = { 1.155, 0.577, -0.577, -1.155, -0.577, 0.577 }
- static const double **hex_dy** [6] = { 0.0, 1.0, 1.0, 0., -1.0, -1.0 }
- static const double **sqr_dx** [4] = { 1.0, -1.0, -1.0, 1.0 }
- static const double **sqr_dy** [4] = { 1.0, 1.0, -1.0, -1.0 }
- static int **ilang** = 0
- static PrimaryId **primaries** [ ]
- static int **neighbours1** [H_MAX_TEL][H_MAX_PIX][H_MAX_NB1]
- static int **nnb1** [H_MAX_TEL][H_MAX_PIX]
- static int **has_nblist** [H_MAX_TEL]
- static int **px_shape_type** [H_MAX_TEL]

7.7.1 Detailed Description

Plot a camera image from H.E.S.S.

/CTA data.

This code is derived from sim_conv2hess.c but now getting the relevant data from the data structure filled after reading the eventio based data, rather than from the internal data structures of sim_hessarray. As a consequence not all information available in the sim_hessarray generated plots is available in the plots generated here. Also some flexibility is lost, concerning for example the pixel shape which is not included in the data.

Author

Konrad Bernloehr

Date

2001 to 2023

7.7.2 Function Documentation

7.7.2.1 camimg_ps_open()

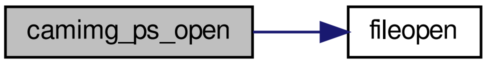
```
static FILE * camimg_ps_open (
            const char * image_fname )  [static]
```

Open the Postscript output file for camera plots.

Keep in mind that we can handle only a single file - which may be closed and re-opended many times.

References fileopen().

Here is the call graph for this function:



7.7.2.2 camimg_ps_pixel_def()

```
static int camimg_ps_pixel_def (
            FILE * psfile,
            CameraSettings * camset,
            int itel,
            double scale,
            double body_diameter )  [static]
```

Define Postscript macros for showing a pixel of the given shape.

Will be renewed for every page.

References H_MAX_TEL, and simtel_camera_settings_struct::size.

7.7.2.3 hesscam_ps_plot()

```
void hesscam_ps_plot (
            const char * image_fname,
            AllHessData * hsdata,
            int itel,
            int type,
            int amp_tm,
            double clip_amp )
```

Write PostScript of camera sum image or sample image to a dedicated file.

Also controlled via environment variables GAMMA_COEFF, GRAY_IMAGE, IMAGE_RANGE, IMAGE_OFFSET for image colors, PLOT_WITH_PIXEL_ID, PLOT_WITH_PIXEL_AMP, PLOT_WITH_PIXEL_PE for overlay text, SHOW_TRUE_PE for showing color for true p.e. number in place of calibrated amplitude.

Parameters

image_fname	The name of the postscript image file. Opened for appending new images.
hsdata	Pointer to the structure containing all data.
itel	The telescope index number.
type	Event type ($<$0: MC events, $>$=0: various type of calib data).
amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak. 3: Show only true p.e. content as amplitude (no samples).
clip_amp	if $>$0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

References simtel_event_data_struct::central, simtel_camera_settings_struct::flen, simtel_central_event_data_↩struct::glob_count, H_MAX_TEL, simtel_tel_event_data_struct::loc_count, simtel_camera_settings_struct::num↩_pixels, simtel_run_header_struct::run, simtel_camera_settings_struct::tel_id, simtel_event_data_struct::teldata, simtel_camera_settings_struct::xpix, and simtel_camera_settings_struct::ypix.

7.7.3 Variable Documentation

7.7.3.1 alt_az_arrow

```
char alt_az_arrow[ ]  [static]
```

Initial value:

```
=
      "n 18000 26000 m "
      "0 100 rl 200 -100 rl -200 -100 rl 0 100 rl -1000 0 rl "
      "cp gs 20 slw black s gr\n"
      "txt5 18700 26100 mtxt (Az) tblack\n"
      "n 17000 25000 m "
      "100 0 rl -100 -200 rl -100 200 rl 100 0 rl 0 1000 rl "
      "cp gs 20 slw black s gr\n"
      "txt5 17000 24600 mtxt (Alt) tblack\n"
      "gs 17800 25500 tr %f rot -17800 -25500 tr\n"
        "n 17800 25500 m "
        "0 100 rl 200 -100 rl -200 -100 rl 0 100 rl -300 0 rl "
        "cp gs 10 slw black s gr\n"
        "txt2 17950 25350 mtxt (y) tblack\n"
        "n 17500 25200 m "
        "100 0 rl -100 -200 rl -100 200 rl 100 0 rl 0 300 rl "
        "cp gs 10 slw black s gr\n"
        "txt2 17700 25200 mtxt (x) tblack\n"
      "gr\n"
```

7.7.3.2 primaries

```
PrimaryId primaries[ ]  [static]
```

Initial value:

```
= {
   {     0, { "gamma",             "Gamma"          } },
   {    -1, { "positron",          "Positron"       } },
   {     1, { "electron",          "Elektron"       } },
   {    -2, { "muon+",             "Myon+"          } },
   {     2, { "muon-",             "Myon-"          } },
   {  -101, { "anti-proton",       "Antiproton"      } },
   {   101, { "proton",            "Proton"          } },
   {   402, { "helium nucleus",    "Heliumkern"      } },
   {  1206, { "carbon nucleus",    "Kohlenstoffkern" } },
   {  1407, { "nitrogen nucles",   "Stickstoffkern"  } },
   {  1608, { "oxygen nucleus",    "Sauerstoffkern"  } },
   {  2412, { "magnesium nucleus", "Magnesiumkern"   } },
   {  2814, { "silicon nucleus",   "Siliziumkern"    } },
   {  5626, { "iron nculeus",      "Eisenkern"       } },
   { 99999, { "type %d",           "Typ %d"          } }
}
```

7.7.3.3 ps_begin_page1

```
char ps_begin_page1[]  [static]
```

Initial value:

```
=
"%%Page: "
```

7.7.3.4 ps_begin_page2

```
char ps_begin_page2[]  [static]
```

Initial value:

```
=
"save\n"
"10 setmiterlimit\n"
"n -1000 31000 m -1000 -1000 l 22000 -1000 l 22000 31000 l cp clip\n"
" 0.02835 0.02835 sc\n"
"gs\n"
"7.500 slw\n"
"black\n"
```

7.7.3.5 ps_end_page

```
char ps_end_page[]  [static]
```

Initial value:

```
=
"gr\n"
"showpage\n"
```

7.7.3.6 ps_head1a

```
char ps_head1a[]  [static]
```

Initial value:

```
=
"%!PS-Adobe-2.0\n"
"%%Title: H.E.S.S. Telescope Simulation"
```

7.7.3.7 ps_head1b

```
char ps_head1b[ ]  [static]
```

Initial value:

```
=
"\n%%Creator:"
```

7.7.3.8 ps_trailer

```
char ps_trailer[ ]  [static]
```

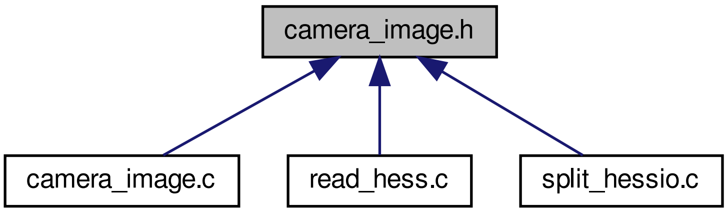
Initial value:

```
=
"rs\n"
```

7.8 camera_image.h File Reference

Function prototypes for camera_image.c.

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



Functions

- void hesscam_ps_plot (const char ∗image_fname, AllHessData ∗hsdata, int itel, int type, int amp_tm, double clip_amp)

    Write PostScript of camera sum image or sample image to a dedicated file.
- void **hesscam_type_sum_plot** (const char ∗image_fname, AllHessData ∗hsdata, int teltype)

7.8.1 Detailed Description

Function prototypes for camera_image.c.

Author

Konrad Bernloehr

Date

2009 to 2018

7.8.2 Function Documentation

7.8.2.1 hesscam_ps_plot()

```
void hesscam_ps_plot (
            const char * image_fname,
            AllHessData * hsdata,
            int itel,
            int type,
            int amp_tm,
            double clip_amp )
```

Write PostScript of camera sum image or sample image to a dedicated file.

Also controlled via environment variables GAMMA_COEFF, GRAY_IMAGE, IMAGE_RANGE, IMAGE_OFFSET for image colors, PLOT_WITH_PIXEL_ID, PLOT_WITH_PIXEL_AMP, PLOT_WITH_PIXEL_PE for overlay text, SHOW_TRUE_PE for showing color for true p.e. number in place of calibrated amplitude.

Parameters

image_fname	The name of the postscript image file. Opened for appending new images.
hsdata	Pointer to the structure containing all data.
itel	The telescope index number.
type	Event type (<0: MC events, >=0: various type of calib data).
amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak. 3: Show only true p.e. content as amplitude (no samples).
clip_amp	if >0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

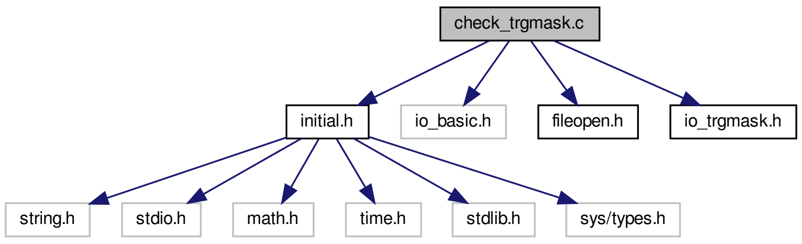
References simtel_event_data_struct::central, simtel_camera_settings_struct::flen, simtel_central_event_data_↩ struct::glob_count, H_MAX_TEL, simtel_tel_event_data_struct::loc_count, simtel_camera_settings_struct::num↩ _pixels, simtel_run_header_struct::run, simtel_camera_settings_struct::tel_id, simtel_event_data_struct::teldata, simtel_camera_settings_struct::xpix, and simtel_camera_settings_struct::ypix.

7.9 check_trgmask.c File Reference

Check consistency of 'trgmask' files produced with gen_trgmask for the CTA prod-2 data sets produced in 2013.

```
#include "initial.h"
#include "io_basic.h"
#include "fileopen.h"
#include "io_trgmask.h"
```

Include dependency graph for check_trgmask.c:



Functions

- int **main** (int argc, char ∗∗argv)

7.9.1 Detailed Description

Check consistency of 'trgmask' files produced with gen_trgmask for the CTA prod-2 data sets produced in 2013.

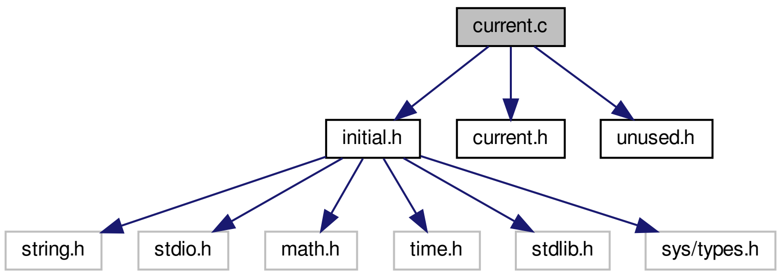
```
Syntax: bin/check_trgmask trgmask-file


@author Konrad Bernloehr
@date 2013 to 2018
```

7.10 current.c File Reference

Code to insert current time string into warnings.

```
#include "initial.h"
#include "current.h"
#include "unused.h"
```

Include dependency graph for current.c:



Macros

- #define **__Current_Module__** 1

Functions

- static long **time_correction** (time_t now)
- time_t current_time ()

  Get the current time in seconds since 1970.0 GMT.
- time_t current_localtime ()

  Like current_time() but should return time in the local time zone.
- void set_current_offset (long off)

  Set current time offset.
- void set_local_offset (long off)

  Set offset of local time zone.
- void reset_local_offset ()

  Reset any previous local time offset.
- static long **time_correction** (_unused_ time_t now)
- char ∗ time_string ()

  Return a pointer to a formatted time-and-date string.
- time_t mkgmtime (struct tm ∗tms)

  Inverse to gmtime() library function.

Variables

- static long **tcor_parm** [3]
- static long **local_offset** = DEFAULT_LOCAL_OFFSET
- static int **local_set** =0

7.10.1 Detailed Description

Code to insert current time string into warnings.

This code is meant for inserting time strings into warnings passed through the code of warning.c. It is not currently used in my code and is not yet multi-threading safe. It is here mainly for improved backward-compatibility with config.c.

Author

Konrad Bernloehr

Date

1995 to 2023

7.10.2 Function Documentation

7.10.2.1 current_localtime()

```
time_t current_localtime (
            void  )
```

Like current_time() but should return time in the local time zone.

The offset of the time zone to GMT must be set by set_local_offset() or it is derived from the machine's internal time zone setup.

Referenced by time_string().

7.10.2.2 current_time()

```
time_t current_time (
            void  )
```

Get the current time in seconds since 1970.0 GMT.

The resulting time includes the last time correction with respect to the server. Therefore, as long as the clock on the local computer is not much slower or faster than the clock on the I/O server, it is the current Greenwich Mean Time on the I/O server.

Returns

Time in seconds since 0h UT on January 1, 1970.

Referenced by push_command_history(), and push_config_history().

7.10.2.3 mkgmtime()

```
time_t mkgmtime (
            struct tm * tms )
```

Inverse to gmtime() library function.

Inverse to gmtime() library function without correction for timezone and daylight saving time.

Parameters

tms	Pointer to time structure as filled by gmtime().

Returns

Time in seconds since 1970.0

7.10.2.4 reset_local_offset()

```
void reset_local_offset (
            void  )
```

Reset any previous local time offset.

Reset any previously set local time offset. The next call to current_localtime() will therefore set the offset to present system value.

Note: in a multi-threaded program this function should be called only at program startup.

Returns

(none)

7.10.2.5 set_current_offset()

```
void set_current_offset (
            long off )
```

Set current time offset.

Set the offset between the time on the time server and the local time (in seconds in the sense 'remote-local').

Note: in a multi-threaded program this function should be called only at program startup.

Parameters

off	Time offset in seconds

Returns

(none)

7.10.2.6 set_local_offset()

```
void set_local_offset (
            long off )
```

Set offset of local time zone.

Set the offset between the local time zone and GMT (in seconds in the sense 'local zone - GMT').

Note: in a multi-threaded program this function should be called only at program startup.

Parameters

off	Time offset in seconds

Returns

(none)

7.10.2.7 time_string()

```
char* time_string (
             void  )
```

Return a pointer to a formatted time-and-date string.

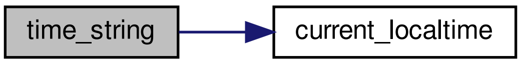
This string is reused (changed) on the next call.

Returns

Time/date character string pointer.

References current_localtime().

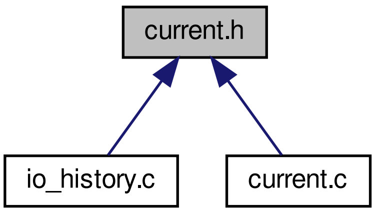
Here is the call graph for this function:



7.11 current.h File Reference

Header file for optional current time add-on to warning.c.

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



Macros

- #define **DEFAULT_LOCAL_OFFSET** 3600

Functions

- time_t current_time (void)

  Get the current time in seconds since 1970.0 GMT.
- time_t current_localtime (void)

  Like current_time() but should return time in the local time zone.
- void set_current_offset (long _toffset)

  Set current time offset.
- void set_local_offset (long _local_offset)

  Set offset of local time zone.
- void reset_local_offset (void)

  Reset any previous local time offset.
- char ∗ time_string (void)

  Return a pointer to a formatted time-and-date string.
- time_t mkgmtime (struct tm ∗tms)

  Inverse to gmtime() library function.

Variables

- time_t **last_data_time**

7.11.1 Detailed Description

Header file for optional current time add-on to warning.c.

Author

Konrad Bernloehr

Date

1993 (original version), 2001, 2007, 2010

7.11.2 Function Documentation

7.11.2.1 current_localtime()

```
time_t current_localtime (
            void )
```

Like current_time() but should return time in the local time zone.

The offset of the time zone to GMT must be set by set_local_offset() or it is derived from the machine's internal time zone setup.

Referenced by time_string().

7.11.2.2 current_time()

```
time_t current_time (
             void  )
```

Get the current time in seconds since 1970.0 GMT.

The resulting time includes the last time correction with respect to the server. Therefore, as long as the clock on the local computer is not much slower or faster than the clock on the I/O server, it is the current Greenwich Mean Time on the I/O server.

Returns

Time in seconds since 0h UT on January 1, 1970.

Referenced by push_command_history(), and push_config_history().

7.11.2.3 mkgmtime()

```
time_t mkgmtime (
             struct tm * tms )
```

Inverse to gmtime() library function.

Inverse to gmtime() library function without correction for timezone and daylight saving time.

Parameters

tms	Pointer to time structure as filled by gmtime().

Returns

Time in seconds since 1970.0

7.11.2.4 reset_local_offset()

```
void reset_local_offset (
             void  )
```

Reset any previous local time offset.

Reset any previously set local time offset. The next call to current_localtime() will therefore set the offset to present system value.

Note: in a multi-threaded program this function should be called only at program startup.

Returns

(none)

7.11.2.5 set_current_offset()

```
void set_current_offset (
             long off )
```

Set current time offset.

Set the offset between the time on the time server and the local time (in seconds in the sense 'remote-local').

Note: in a multi-threaded program this function should be called only at program startup.

Parameters

off	Time offset in seconds

Returns

(none)

7.11.2.6 set_local_offset()

```
void set_local_offset (
             long off )
```

Set offset of local time zone.

Set the offset between the local time zone and GMT (in seconds in the sense 'local zone - GMT').

Note: in a multi-threaded program this function should be called only at program startup.

Parameters

off	Time offset in seconds

Returns

(none)

7.11.2.7 time_string()

```
char* time_string (
             void )
```

Return a pointer to a formatted time-and-date string.

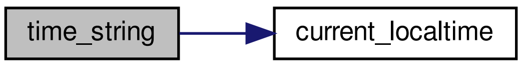
This string is reused (changed) on the next call.

Returns

Time/date character string pointer.

References current_localtime().

Here is the call graph for this function:

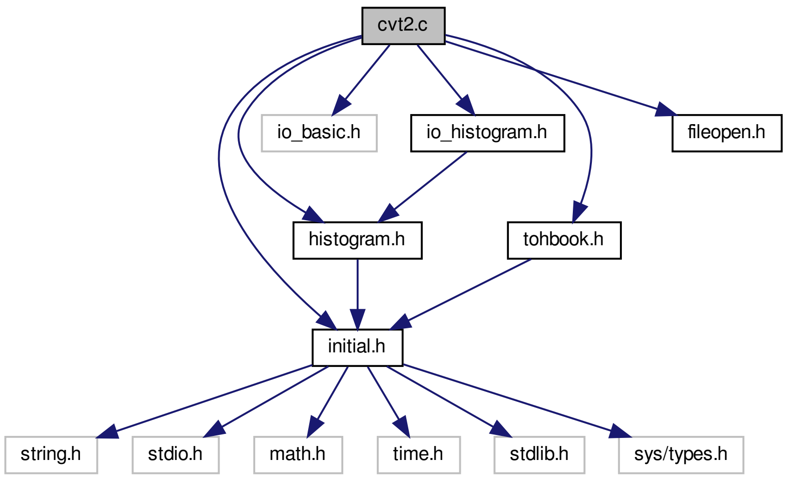


7.12 cvt2.c File Reference

Utility program for converting histograms to HBOOK format.

```
#include "initial.h"
#include "histogram.h"
#include "io_basic.h"
#include "tohbook.h"
#include "io_histogram.h"
#include "fileopen.h"
```

Include dependency graph for cvt2.c:



Functions

- int main (int argc, char ∗∗argv)

  Main program.

7.12.1 Detailed Description

Utility program for converting histograms to HBOOK format.

```
Syntax:  hdata2hbook [ input_file [ output_file ] ]
    or:  hdata2hbook -a input_files ... -o output_file
```

The program was originally called `cvt2`. The default input file name is 'testpattern.hdata', the default output file name is 'testpattern.hbook' or the input file name with extension '.hbook' (instead of '.hdata'). The histograms may be within multiple I/O blocks of the input file. Only non-empty histograms are written to output.

With the '-a' option, all identical histograms in the input files will be added up before writing them to output.

Author

Konrad Bernloehr

Date

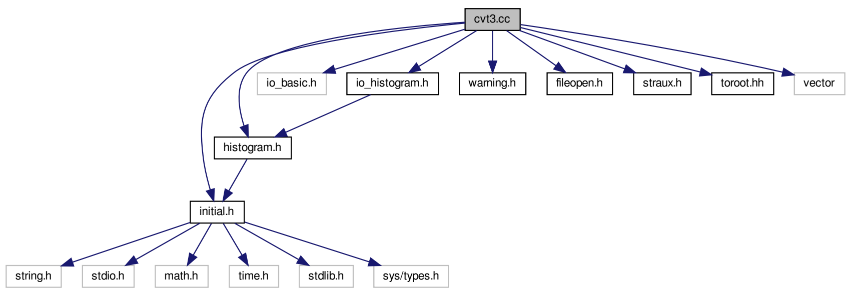
2001 to 2014

7.13 cvt3.cc File Reference

Conversion of eventio histograms to ROOT format.

```
#include "initial.h"
#include "histogram.h"
#include "io_basic.h"
#include "io_histogram.h"
#include "warning.h"
#include "fileopen.h"
#include "straux.h"
#include "toroot.hh"
#include <vector>
```

Include dependency graph for cvt3.cc:



Functions

- int **read_file** (IO_BUFFER ∗iobuf, const char ∗fname, int add_flag, int list_flag)
- int **main** (int argc, char ∗∗argv)

7.13.1 Detailed Description

Conversion of eventio histograms to ROOT format.

```
   Syntax:  hdata2root [ input_file [ output_file ] ]
       or:  hdata2root -a input_files ... -o output_file
```

The program was originally called `cvt3`. The default input file name is 'testpattern.hdata', the default output file name is 'testpattern.root' or the input file name with extension '.root' (instead of '.hdata'). The histograms may be within multiple I/O blocks of the input file. Only non-empty histograms are written to output. Take care not to replace any ROOT data format you wanted to keep.

With the '-a' option, all identical histograms in the input files will be added up before writing them to output.

Author

Konrad Bernloehr

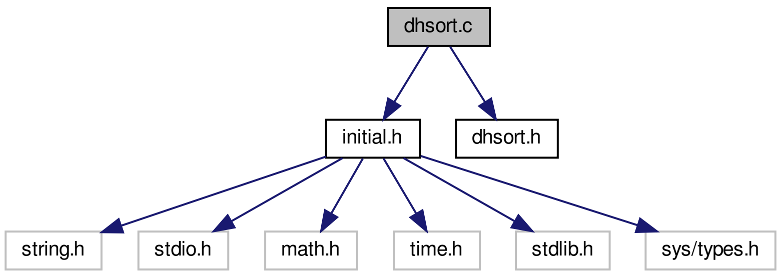
Date

2002 to 2022

7.14 dhsort.c File Reference

dhsort - double type number heapsort

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



Functions

- void dhsort (double ∗dnum, int nel)

  Perform a heap sort on a double array starting at dnum.

7.14.1 Detailed Description

dhsort - double type number heapsort

Author

Konrad Bernloehr

Date

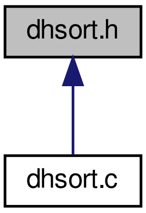
1997 to 2018

```
        Based on algorithms by Jon Bentley [Communications of the ACM v
        28 n 3 p 245 (Mar 85) and v 28 n 5 p 456 (May 85)], and the
        sort interface routines by Allen I.  Holub [Dr.  Dobb's Journal
        #102 (Apr 85)].

Notes...
        This routine sorts N doubles in worst-case time proportional to
        N*log(N).  The heapsort was discovered by J.  W.  J.  Williams
        [Communications of the ACM v 7 p 347-348 (1964)] and is
        discussed by D.  E.  Knuth [The Art of Computer Programming,
        Volume 3: Sorting and Searching, Addison-Wesley, Reading,
        Mass., 1973, section 5.2.3].

        This algorithm depends on a portion of an array having the
        "heap" property.  The array X has the property heap[L,U] if:

                for all      L, i, and U
                such that    2L <= i <= U
                we have      X[i div 2] <= X[i]
```

7.15 dhsort.h File Reference

Function prototypes for dhsort.c.

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



Functions

- void dhsort (double *dnum, int nel)

  Perform a heap sort on a double array starting at dnum.

7.15.1 Detailed Description

Function prototypes for dhsort.c.

Author

Konrad Bernloehr

Date

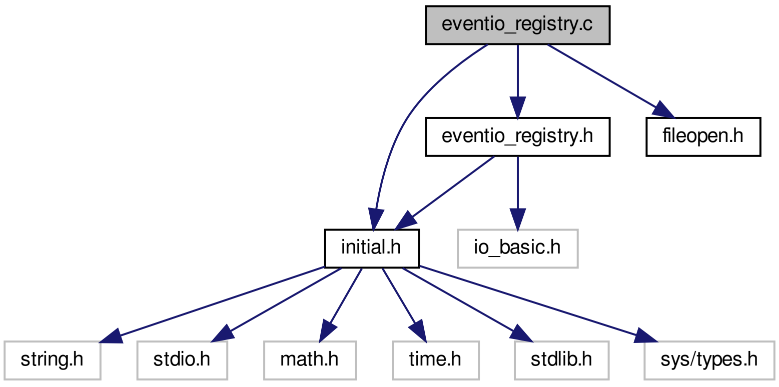
1997 to 2018

7.16 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 ∗ new_reg_entry (unsigned long t, const char ∗n, const char ∗d)

  Allocate a new entry for the registry.
- int read_eventio_registry (const char ∗fname)

  Read the type names and descriptions into the registry.
- static void read_default_registry (void)

  By default the registry contents will be searched in a few places.
- struct ev_reg_entry ∗ find_ev_reg_std (unsigned long t)

  Find an entry for a given type number in the registry.
- void set_ev_reg_std ()

  Set the default registry search function.

Variables

- static struct ev_reg_chain ∗ **ev_reg_start** = NULL

7.16.1 Detailed Description

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

Author

Konrad Bernloehr

Date

2014 to 2018

7.16.2 Function Documentation

7.16.2.1 find_ev_reg_std()

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

Find an entry for a given type number in the registry.

This is the standard implementation being used by default where available.

Referenced by set_ev_reg_std().

7.16.2.2 read_eventio_registry()

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

Read the type names and descriptions into the registry.

Note: this will only be done once.

Referenced by read_default_registry().

7.16.2.3 set_ev_reg_std()

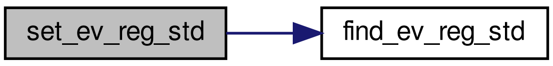
```
void set_ev_reg_std (
            void  )
```

Set the default registry search function.

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

References find_ev_reg_std().

Here is the call graph for this function:

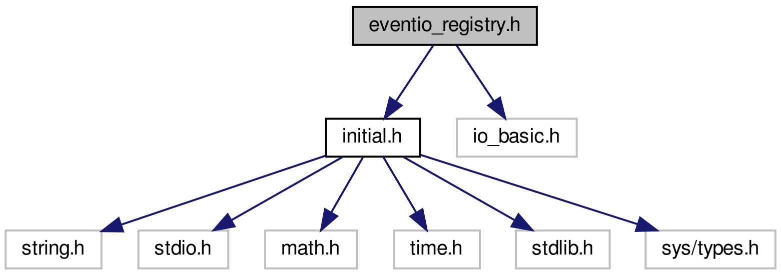


7.17 eventio_registry.h File Reference

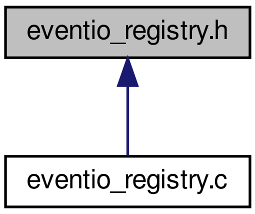
Register and enquire about well-known I/O block types.

```
#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

- int read_eventio_registry (const char ∗fname)

    Read the type names and descriptions into the registry.
- struct ev_reg_entry ∗ find_ev_reg_std (unsigned long t)

    Find an entry for a given type number in the registry.
- void set_ev_reg_std (void)

    Set the default registry search function.

7.17.1 Detailed Description

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

Author

Konrad Bernloehr

Date

2014

7.17.2 Function Documentation

7.17.2.1 find_ev_reg_std()

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

Find an entry for a given type number in the registry.

This is the standard implementation being used by default where available.

Referenced by set_ev_reg_std().

7.17.2.2 read_eventio_registry()

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

Read the type names and descriptions into the registry.

Note: this will only be done once.

Referenced by read_default_registry().

7.17.2.3 set_ev_reg_std()

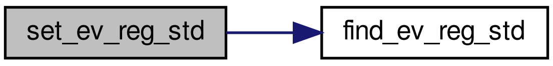
```
void set_ev_reg_std (
            void  )
```

Set the default registry search function.

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

References find_ev_reg_std().

Here is the call graph for this function:



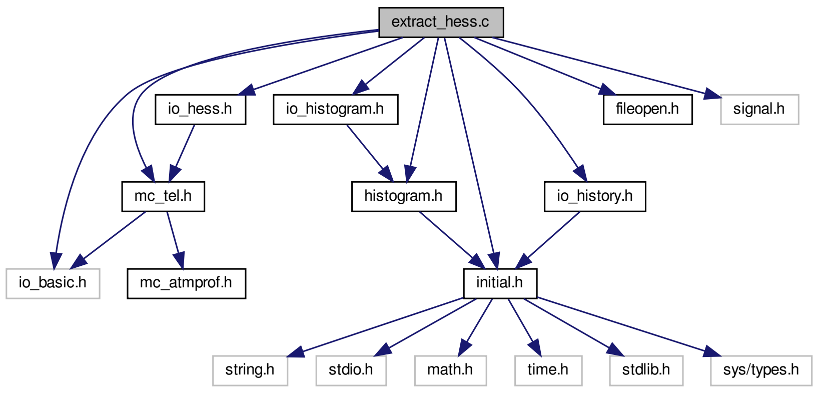
7.18 extract_hess.c File Reference

Extract simulated calibration type event data originally encapsulated.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_history.h"
#include "io_hess.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
```

```
#include <signal.h>
```

Include dependency graph for extract_hess.c:



Functions

- static void syntax (char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**

7.18.1 Detailed Description

Extract simulated calibration type event data originally encapsulated.

Author

Konrad Bernloehr

Date

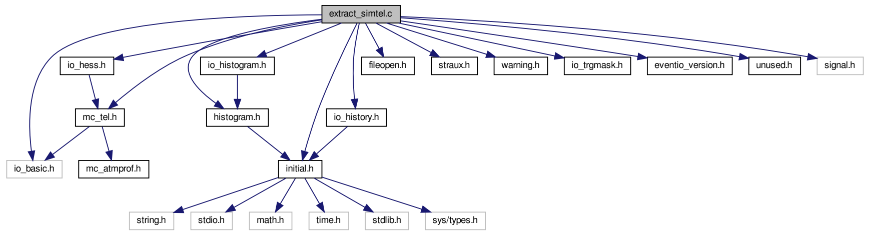
2003 to 2022

7.19 extract_simtel.c File Reference

A program for extracting data for a subset of simulated telescopes.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_history.h"
#include "io_hess.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
#include "straux.h"
#include "warning.h"
#include "io_trgmask.h"
#include "eventio_version.h"
#include "unused.h"
#include <signal.h>
```

Include dependency graph for extract_simtel.c:



Data Structures

- struct map_tel_struct

  Structure with per output telescope information keeping track of prerequisites.

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void syntax (const char ∗program)

  Show program syntax.
- int find_in_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the input structures.
- int find_out_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the output structures.
- int find_mapped_telescope (int tel_id, int ifile)

  Mapping from telescope ID on input to telescope ID on output, with check.
- int write_io_block_to_file (IO_BUFFER ∗iobuf, FILE ∗f)

  Write an I/O block as-is to another file than foreseen for the I/O buffer.
- int **check_for_delayed_write** (IO_ITEM_HEADER ∗item_header, _unused_ int ifile, AllHessData ∗hsdata↩
  _out, IO_BUFFER ∗iobuf_out)

- int merge_data_from_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header, int ifile, AllHessData ∗hsdata, AllHessData ∗hsdata_out, IO_BUFFER ∗iobuf_out)

    Processing of I/O blocks from the input file.
- int check_autoload_trgmask (const char ∗input_fname, IO_BUFFER ∗iobuf, int ifile)

    Check for a 'trgmask.gz' file matching the given input data file name and, if it exists, extract the corrected trigger bit patterns from it.
- void **print_process_status** (int prev_type1, int this_type1)
- int **read_map** (const char ∗map_fname)
- int main (int argc, char ∗∗argv)

    Main program.

Variables

- static int **interrupted**
- static int **verbose** = 0
- struct map_tel_struct **map_tel** [H_MAX_TEL]
- int map_to [2][H_MAX_TEL+1]

    Mapping structures from input telescope ID to output telescope ID.
- int tel_idx [2][H_MAX_TEL+1]

    Mapping from telescope IDs to offsets in the data structures, first for input telescope IDs.
- int tel_idx_out [H_MAX_TEL+1]

    Mapping from output telescope ID to offset in output data structures.
- int **ntel1**
- int **ntel2**
- int **ntel**
- int **nrtel1**
- int **nrtel2**
- long **event1** = -1
- long **event2** = 0
- long **ev_hess_event** = 0
- long ev_pe_sum = 0

    For delayed writing.
- int **run1** = -1
- int **run2** = -1
- int **min_trg** = 2
- static struct trgmask_set ∗ **tms** [2] = { NULL, NULL }
- static struct trgmask_hash_set ∗ **ths** [2] = { NULL, NULL }
- static int **events** [2] = { 0, 0 }
- static int **mcshowers** [2] = { 0, 0 }
- static int **mcevents** [2] = { 0, 0 }
- static int **max_list** = 999

7.19.1 Detailed Description

A program for extracting data for a subset of simulated telescopes.

The program will read sim_telarray raw or DST data from one input file, map telescope ID according to how they appear in the list of selected telescopes and write the re-mapped blocks to an output file. It behaves basically like 'merge_simtel' with only one input file.

Inputs expected - and the action to be performed: Type Once per run: 70 (history) - Write as-is, no attempt to identify which part is relevant for which telescope 2000 (run_header) - Re-write as needed for telescope list and positions

2001 (MC run header) - Write as-is, nothing telescope-specific 1212 (input config = CORSIKA inputs) - Write as-is, nothing telescope-specific 1216 (atmospheric density profile) - Only one needed (should be identical, duplicate) Once per telescope (and per run for raw & DST levels 0-2; just once for DST level 3): 75 (metaparam) - Write after mapping of telescope ID (if mapped); global remains ID -1. 2002 (camera settings) - Write after mapping of telescope ID (if mapped) 2003 (camera organization) - Write after mapping of telescope ID (if mapped) 2004 (pixel settings) - Write after mapping of telescope ID (if mapped) 2005 (pixel disable) - Write after mapping of telescope ID (if mapped) 2006 (camera software settings) - Write after mapping of telescope ID (if mapped) 2008 (tracking settings) - Write after mapping of telescope ID (if mapped) 2007 (pointing corrections) - Write after mapping of telescope ID (if mapped) 2022 (telescope monitoring) - Write after mapping of telescope ID (if mapped) 2023 (Laser calibration) - Write after mapping of telescope ID (if mapped) 2033 (MC pixel monitoring) - Write after mapping of telescope ID (if mapped) Per shower: once: 2020 (MC shower) - Write as-is, nothing telescope-specific per array: 2021 (MC event) - Write as-is, nothing telescope-specific Optional per event; not immediately written but delayed until next MC etc. block: 2026 (MC pe sum) - ??? 1204 (photo-electrons individually) - ??? 2010 (event) - Needs remapping at all levels At end of run: 2024 (run statistics - usually not present) 2025 (MC run statistics - usually not present) 100 (histograms) - Cannot be remapped properly (but few histograms are telescope-specific)

Note: Ignoring 'trgmask' files - these are not relevant any more.

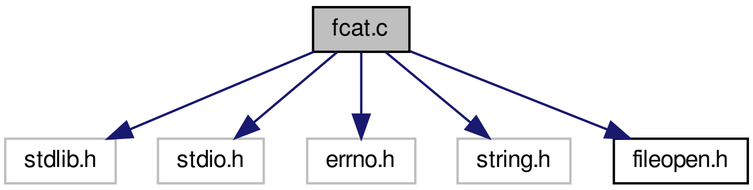
```
A program for extracting data for a subset of simulated telescopes.

Syntax:  extract_simtel [ options ] input output
Options:
     --map-file       : Load the telescope ID mapping from a file.
     --only-telescope: List of telescopes on command line rather than map file.
     --auto-trgmask   : Load trgmask.gz files for each input file where available.
     --min-trg-tel n : Require at least n telescopes in extracted event (default: 2).
     --verbose        : Show events being extracted.


@author  Konrad Bernloehr
@date    2015 to 2023
```

7.20 fcat.c File Reference

Trivial test and utility program for the fileopen/fileclose functions.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include "fileopen.h"
```

Include dependency graph for fcat.c:



Macros

- #define **BSIZE** 8192

Functions

- void **syntax** (void)
- int **main** (int argc, char ∗∗argv)

7.20.1 Detailed Description

Trivial test and utility program for the fileopen/fileclose functions.

Author

Konrad Bernloehr

Date

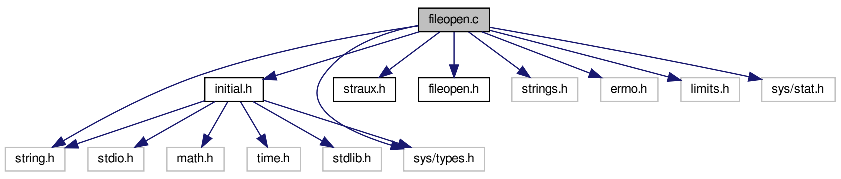
2010 to 2018

7.21 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 <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
```

Include dependency graph for fileopen.c:



Data Structures

- struct rep_entry

Macros

- #define **PATH_MAX** 4096

Functions

- static void fileopen_print_report ()

  Function called at program exit to report all names of files opened through fileopen().
- static void fileopen_env_init (void)

  Initialize internal variables from environment on first call to fileopen().
- static void fileopen_add_report (const char ∗fname, const char ∗mode)

  Add a filename to the list of filenames reported at program end into ".fileopen.lis" or FILEOPEN_LIST.
- static FILE ∗ popenx (const char ∗fname, const char ∗mode)

  Function called by fileopen() to actually open a pipe, including files compressed through external tools.
- static FILE ∗ fopenx (const char ∗fname, const char ∗mode)

  Function called by fileopen() to actually open a plain file.
- void set_permissive_pipes (int p)

  Enable or disable the permissive execution of pipes.
- void enable_permissive_pipes ()

  Enable the permissive execution of pipes.
- void disable_permissive_pipes ()

  Disable the permissive execution of pipes.
- struct incpath ∗ **get_include_path** (void)
- static void freepath ()

  Free a whole list of include path elements.
- static void freeexepath ()

  Free a whole list of execution path elements.
- void initpath (const char ∗default_path)

  Init the path list, with default_path as the only entry.
- void **initexepath** (const char ∗default_exe_path)
- void listpath (char ∗buffer, size_t bufsize)

  Show the list of include paths.
- void addpath (const char ∗name)

  Add a path to the list of include paths, if not already there.
- void addexepath (const char ∗name)

  Add a path to the list of execution paths, if not already there.
- static FILE ∗ exe_popen (const char ∗fname, const char ∗mode)

  Helper function for opening a pipe from or to a given program.
- static FILE ∗ cmp_popen (const char ∗fname, const char ∗mode, int compression)

  Helper function for opening a compressed file through a fifo.
- static FILE ∗ uri_popen (const char ∗fname, const char ∗mode, int compression)

  Helper function for opening a file with a URI (`http://` *etc.).*
- static FILE ∗ ssh_popen (const char ∗fname, const char ∗mode, int compression)

  Helper function for opening a file on a remote SSH server.
- FILE ∗ fileopen (const char ∗fname, const char ∗mode)

  Search for a file in the include path list and open it if possible.
- int fileclose (FILE ∗f)

  Close a file or fifo but not if it is one of the standard streams.

Variables

- static int verbose = 0

  Use to decide if open/close success/failure is reported.
- static int **parallel** = 0
- static int **report** = 0
- static int **with_fallback** = 1
- static int **with_exec** = 1
- static const char ∗ **fileopen_list** = ".fileopen.lis"
- static int **foei_done** = 0
- struct rep_entry **rep_base**
- static struct incpath ∗ root_path = NULL

  The starting element of include paths.
- static struct incpath ∗ root_exe_path = NULL

  The starting element for execution paths.
- static int permissive_pipes = 0

  Allow any execution pipe command if this variable is non-zero.

7.21.1 Detailed Description

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

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 to 2023

7.21.2 Function Documentation

7.21.2.1 addexepath()

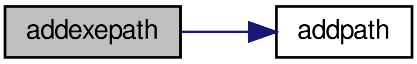
```
void addexepath (
            const char * name )
```

Add a path to the list of execution paths, if not already there.

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.

Here is the call graph for this function:



7.21.2.2 addpath()

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

Add a path to the list of include paths, if not already there.

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 incpath::path.

Referenced by addexepath().

7.21.2.3 fileopen_add_report()

```
static void fileopen_add_report (
            const char * fname,
            const char * mode )  [static]
```

Add a filename to the list of filenames reported at program end into ".fileopen.lis" or FILEOPEN_LIST.

Compressed files are reported both with their real file name and with the pipe calling the external tool handling the compression.

7.21.2.4 fileopen_env_init()

```
static void fileopen_env_init (
            void  )  [static]
```

Initialize internal variables from environment on first call to fileopen().

Environment variables recognized:

- FILEOPEN_VERBOSE
- FILEOPEN_PARALLEL
- FILEOPEN_NO_FALLBACK
- FILEOPEN_NO_EXEC
- FILEOPEN_REPORT
- FILEOPEN_LIST

References verbose.

7.21.2.5 fileopen_print_report()

```
static void fileopen_print_report (
            void  )  [static]
```

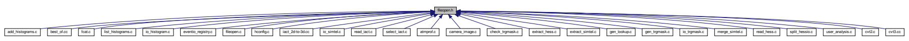
Function called at program exit to report all names of files opened through fileopen().

The value of the FILEOPEN_REPORT environment variable can be used to select format options (1...8), 0=off. The simplest format is for FILEOPEN_REPORT=1, showing just the file names. For other values, the name of the program executed is included and the (first-time) mode and/or the number of times the file was opened may be included. The output is appended to file '.fileopen.lis' or the file name given in environment variable FILEOPEN_LIST

7.22 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 initpath (const char ∗default_path)

  Init the path list, with default_path as the only entry.
- void **initexepath** (const char ∗default_path)
- void listpath (char ∗buffer, size_t bufsize)

  Show the list of include paths.
- void addpath (const char ∗name)

  Add a path to the list of include paths, if not already there.
- void addexepath (const char ∗name)

  Add a path to the list of execution paths, if not already there.
- FILE ∗ fileopen (const char ∗fname, const char ∗mode)

  Search for a file in the include path list and open it if possible.
- int fileclose (FILE ∗f)

  Close a file or fifo but not if it is one of the standard streams.
- struct incpath ∗ **get_include_path** (void)
- void set_permissive_pipes (int p)

  Enable or disable the permissive execution of pipes.
- void enable_permissive_pipes (void)

  Enable the permissive execution of pipes.
- void disable_permissive_pipes (void)

  Disable the permissive execution of pipes.

7.22.1 Detailed Description

Function prototypes for fileopen.c.

Author

Konrad Bernloehr

Date

2000 to 2019

7.22.2 Function Documentation

7.22.2.1 addexepath()

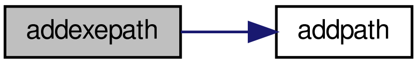
```
void addexepath (
             const char * name )
```

Add a path to the list of execution paths, if not already there.

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.

Here is the call graph for this function:



7.22.2.2 addpath()

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

Add a path to the list of include paths, if not already there.

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").

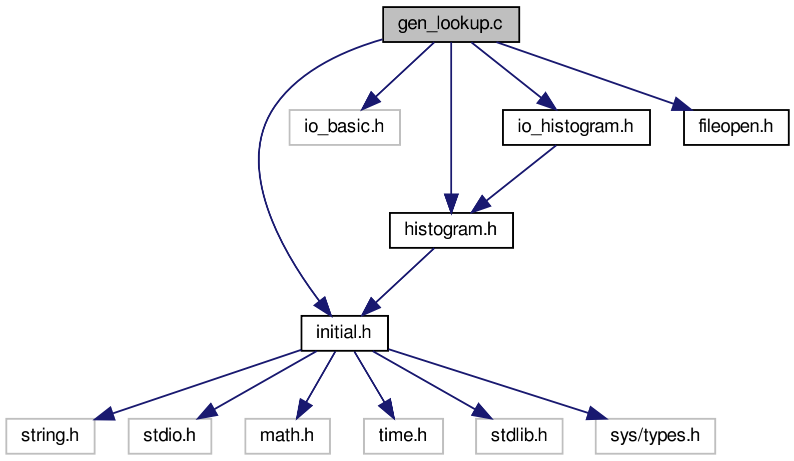
References incpath::path.

Referenced by addexepath().

7.23 gen_lookup.c File Reference

Generate image shape and energy lookups for user analysis in read_hess.

```
#include "initial.h"
#include "io_basic.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
```
Include dependency graph for gen_lookup.c:



Functions

- void fill_gaps ()

  Fill gaps in those histograms used for generating the lookups.
- void gen_image_lookups ()

  Generate the lookups for image shape parameters and energy.
- void **fill_ebias_correction** (void)
- void **syntax** (char ∗prgm)
- int **main** (int argc, char ∗∗argv)

Variables

- HISTOGRAM ∗ **h18000**
- HISTOGRAM ∗ **h18001**
- HISTOGRAM ∗ **h18011**
- HISTOGRAM ∗ **h18012**
- HISTOGRAM ∗ **h18021**
- HISTOGRAM ∗ **h18022**
- HISTOGRAM ∗ **h18051**
- HISTOGRAM ∗ **h18052**
- HISTOGRAM ∗ **h18100**
- HISTOGRAM ∗ **h18101**

- HISTOGRAM ∗ **h18111**
- HISTOGRAM ∗ **h18112**
- HISTOGRAM ∗ **h18121**
- HISTOGRAM ∗ **h18122**
- HISTOGRAM ∗ **h18151**
- HISTOGRAM ∗ **h18152**
- HISTOGRAM ∗ **h18113**
- HISTOGRAM ∗ **h18114**
- HISTOGRAM ∗ **h18123**
- HISTOGRAM ∗ **h18124**
- HISTOGRAM ∗ **h18140**
- HISTOGRAM ∗ **h18141**
- HISTOGRAM ∗ **h18153**
- HISTOGRAM ∗ **h18154**
- HISTOGRAM ∗ **h18005**
- HISTOGRAM ∗ **h18006**
- HISTOGRAM ∗ **h18071**
- HISTOGRAM ∗ **h18072**
- HISTOGRAM ∗ **h18081**
- HISTOGRAM ∗ **h18082**
- HISTOGRAM ∗ **h18105**
- HISTOGRAM ∗ **h18106**
- HISTOGRAM ∗ **h18171**
- HISTOGRAM ∗ **h18172**
- HISTOGRAM ∗ **h18181**
- HISTOGRAM ∗ **h18182**
- HISTOGRAM ∗ **h18173**
- HISTOGRAM ∗ **h18174**
- HISTOGRAM ∗ **h18183**
- HISTOGRAM ∗ **h18184**
- HISTOGRAM ∗ **h18200**
- HISTOGRAM ∗ **h18201**
- HISTOGRAM ∗ **h18211**
- HISTOGRAM ∗ **h18212**
- HISTOGRAM ∗ **h18301**
- HISTOGRAM ∗ **h18311**
- HISTOGRAM ∗ **h18321**
- HISTOGRAM ∗ **h18322**

7.23.1 Detailed Description

Generate image shape and energy lookups for user analysis in read_hess.

Read_hess must be run with user analysis once and the generated histogram file is used by this program to generate the lookups. The lookup file is used in the next round of read_hess user analysis, if found under the desired name. Look at the last lines of output from read_hess (or at the beginning, right after the history) to see how the lookup file should be called (depends on tail cut parameters, and so on).

Author

Konrad Bernloehr

Date

2006 to 2022

7.23.2 Function Documentation

7.23.2.1 fill_gaps()

```
void fill_gaps (
             void  )
```

Fill gaps in those histograms used for generating the lookups.

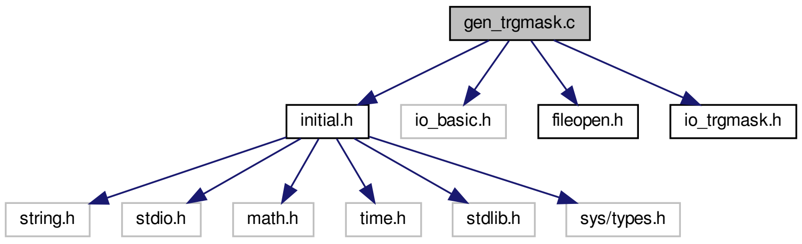
Depending on the physical quantities we have different strategies for interpolation/extrapolation/smoothing.

7.24 gen_trgmask.c File Reference

A utility program for fixing problems with simulation data which does not have the correct bit pattern of telescope triggers but the correct pattern can be extracted from the log files.

```
#include "initial.h"
#include "io_basic.h"
#include "fileopen.h"
#include "io_trgmask.h"
```

Include dependency graph for gen_trgmask.c:



Functions

- void **syntax** (char ∗prgname)
- int **main** (int argc, char ∗∗argv)

7.24.1 Detailed Description

A utility program for fixing problems with simulation data which does not have the correct bit pattern of telescope triggers but the correct pattern can be extracted from the log files.

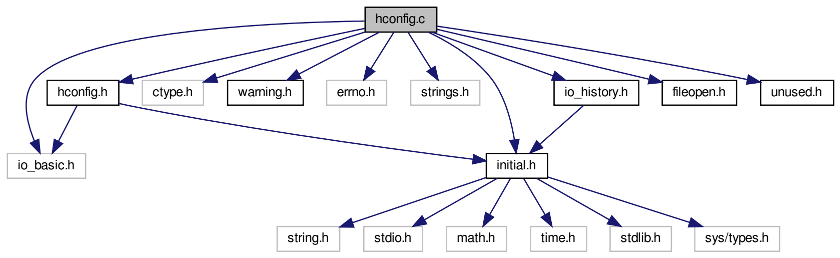
```
Syntax: bin/gen_trgmask log-file [ trgmask-file ]
    or: bin/gen_trgmask -l  trgmask-file
The first variant will create a file with a single data block
for the trigger mask patterns recovered from the log file.
The default file name is derived with extension .trgmask.gz
Note that only data for one run per file is supported.
The second variant will list the contents of such a file.

@author Konrad Bernloehr
@date   2013 to 2018
```

7.25 hconfig.c File Reference

Configuration control and procedure call interface.

```
#include "initial.h"
#include "io_basic.h"
#include <ctype.h>
#include "warning.h"
#include <errno.h>
#include <strings.h>
#include "hconfig.h"
#include "io_history.h"
#include "fileopen.h"
#include "unused.h"
```

Include dependency graph for hconfig.c:



Data Structures

- struct ConfigBlockStruct

  Configuration is organized in sections.
- struct config_specific_data
- struct Binary_Interface_Chain

Macros

- #define **get_config_specific**() (&config_defaults)
- #define **TMP_FORMAT** "cfg%d.tmp"

Typedefs

- typedef struct ConfigBlockStruct **CONFIG_BLOCK**

Functions

- static int do_config (CONFIG_ITEM ∗item, CONST char ∗line)

  Internal configuration function.
- static void **config_syntax_error** (const char ∗name, const char ∗text)
- static void **config_info** (const char ∗name, const char ∗text)
- static int set_config_values (CONFIG_ITEM ∗item, int first, int last, char ∗text)

  Set configuration values (internal usage only).
- static void display_config_initial (CONFIG_ITEM ∗item)

  Display initial values of a single configuration item (internal usage only).
- static void display_config_current (CONFIG_ITEM ∗item)

  Display current values of a single configuration item (internal usage only).
- static void display_config_item (CONFIG_ITEM ∗item)

  Display a single configuration item (internal usage only).
- static int **do_reset_func** (const char ∗text)
- static int **signed_config_val** (const char ∗name, const char ∗text, const char ∗lbound, const char ∗ubound, int strict, long ∗ival)
- static int **unsigned_config_val** (const char ∗name, const char ∗text, const char ∗lbound, const char ∗ubound, int strict, unsigned long ∗uval)
- static int **hex_config_val** (const char ∗name, const char ∗text, const char ∗lbound, const char ∗ubound, int strict, unsigned long ∗uval)
- static int **real_config_val** (const char ∗name, const char ∗text, const char ∗lbound, const char ∗ubound, int strict, double ∗rval)
- static int **bool_config_val** (const char ∗name, const char ∗text, const char ∗lbound, const char ∗ubound, int strict, bool ∗bval)
- static bool **read_boolean** (const char ∗text)
- static int **f_show_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_lock_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_unlock_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_limit_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_status_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_list_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_initlist_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_typelist_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_get_config** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_echo** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_warning** (const char ∗name, CONFIG_VALUES ∗val)
- static int **f_error** (const char ∗name, CONFIG_VALUES ∗val)
- static int **save_config_values** (CONFIG_ITEM ∗item, int first, int last)
- static int **restore_config_values** (CONFIG_ITEM ∗item, int first, int last)
- int build_config (CONFIG_ITEM ∗items, const char ∗section)

  Build up the configuration by adding another section of configuration definitions.
- int init_config (char ∗(∗fptr)(void))

  Initialize the configuration after all build_config() calls.
- void unhook_internal ()

  Disable access to internal functions via configuration.
- void rehook_internal ()

  Enable access again to internal functions via configuration.
- int reload_config (char ∗(∗fptr)(void))

  Reload some configuration using the file name/preprocessor as set up for init_config() or with different file etc.
- CONFIG_ITEM ∗ find_config_item (const char ∗name)

  Find a configuration item by its name (mainly for internal usage).
- int **verify_config_section** (char ∗section)

- int set_config_history (PFITI fptr)

  Set a function for recording the history of the configuration settings.
- int reconfig (char ∗text)

  Modify the configuration after init_config() has been called.
- static int **lock_unlock_status** (const char ∗name, int lock)
- static int **f_lock_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_unlock_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_limit_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_status_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_list_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_initlist_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_typelist_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_get_config** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_echo** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_warning** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int **f_error** (const char ∗name, _unused_ CONFIG_VALUES ∗dummy)
- static int f_show_config (const char ∗name, _unused_ CONFIG_VALUES ∗val)

  Display the current configuration status (internal usage only).
- static int **t_add_to_text** (const char ∗txt, char ∗buf, size_t lbuf, size_t ∗plcur)
- const char ∗ get_config_current (CONFIG_ITEM ∗item, char ∗buf, size_t lbuf)

  Retrieve current settings of a single configuration item as a text string.
- int **is_signed_number** (const char ∗text)
- int **is_unsigned_number** (const char ∗text)
- int **is_hex_number** (const char ∗text)
- int **is_bin_number** (const char ∗text)
- unsigned long **decode_bin_number** (const char ∗text)
- int **is_real_number** (const char ∗text)
- int **is_boolean** (const char ∗text)
- void set_config_filename (const char ∗fname)

  Set the name of the configuration file to be read by the function read_config_lines().
- char ∗ get_config_filename ()

  Return the current value of the configuration file name.
- void set_config_preprocessor (char ∗preproc)

  Set the command name and options of a preprocessor for configuration files to be read by function read_config_lines().
- char ∗ get_config_preprocessor ()

  Return the current value of the configuration preprocessor.
- void set_config_stack (char ∗∗stack)

  Set a list of configuration lines to be processed before any lines from a file are read by read_config_lines().
- char ∗ read_config_lines ()

  Read configuration data from a file and return it line by line to the calling function (one line per call).
- int read_config_status ()

  Return the status of reading a configuration file with read_config_lines() in a preceding call to init_config().
- int define_config_binary_interface (int item_type, size_t elem_size, void ∗(∗new_func)(int nelem, int item←_type), int(∗delete_func)(void ∗ptr, int nelem, int item_type), int(∗read_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type), int(∗write_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type), int(∗readtext_←func)(void ∗bin_item, char ∗text, int item_type), int(∗list_func)(void ∗bin_item, int item_type), int(∗copy_←func)(void ∗bin_item_to, void ∗bin_item_from, int io_type))

  Define a binary interface for an I/O type.
- struct Config_Binary_Item_Interface ∗ find_config_binary_interface (int item_type)

  Find the matching binary interface for given item type.

Variables

- static CONFIG_ITEM default_config [ ]

  Internal functions of the hconfig package.
- static CONFIG_BLOCK **first_config_block**
- static int **internal_unhooked** = 0
- static PFITI **history_function**
- int **config_level**
- static struct config_specific_data **config_defaults**
- static struct Binary_Interface_Chain ∗ **bin_chain_root**
- static char **cfg_fname** [1024]
- static char **preprocessor** [4096] = ""
- static char ∗∗ **cfg_stack**
- static int **read_status**

7.25.1 Detailed Description

Configuration control and procedure call interface.

Author

Konrad Bernloehr

Date

2001 to 2023

```
This is the module controlling all configuration except
that a function has to be supplied that collects input line
for line. Most functions in this file are for internal use only
and are given a 'static' modifier. The only functions to be
called by the user are


       build_config()
       init_config()
       reconfig().







In order to set up the configuration, one or several calls to
build_config() should be done, each with a list of 'configuration
items' ('CONFIG_ITEM *items') terminated by a NULL_CONFIG_ITEM
as an end marker. The list must be of 'static' or global/'extern'
type and none of its entries must be modified by the user in any
way, once they have been passed to build_config.

Such a list might look like the following example:


       static CONFIG_ITEM cfg_list[] =
       {
          { "ANY_Numbers", "Int", 30, iarray },
          { "ANY_Function", "function", -1, NULL, some_function },
          { "REAL_Number", "R", 10, dblarray, NULL, "0-9: 99.9",
              "10", "100", CFG_REQUIRE_ALL_DATA | CFG_REJECT_MODIFICATION },
          { "DYnAllocArray", "i", 100, NULL, NULL },
          { NULL_CONFIG_ITEM }
       }
```

```
The components of each item are:


        1) The name, consisting of letters, digits, and '_'.
           In external data the items are referenced by their name
           which may be abbreviated and is case-insensitive. However,
           the name used for the definition is case-sensitive in the
           current implementation. The first lowercase letter indicates
           the minimum length of accepted abbrevations. In the example
           above "ANY_Numbers" may be abbreviated as "any_n", "any_nu",
           and so on, "DYnAllocArray" as "dy", "dyn", and so on.
           It is the user's responsibility the avoid conflicts of the
           accepted abbreviations of any two items.
        2) The type which may be an abbreviation of one of the following:
              "Character", "Short", "Integer", "Long" (signed integer types),
              "UCharacter", "UShort", "UInteger, "ULong" (unsigned types)
              "FLoat", "Real", "Double" (floating point, "Real" == "Double"),
              "Bool" (accepts value like "true", "OFF", ..., in addition to
                  0/1 but rejects any other integers than 0 and 1),
              "IBool" (same as "Bool" but storing in an integer),
              "UBool" (same again, into an unsigned int),
              "Text" (simple text, character string),
              "FUnction" (a function reference, not a data reference).
        3) The number of data element. Must be -1 for "FUnction" type.
           The terminating '\0' in characters strings should be included.
        4) A data pointer of any type. Must be NULL for "FUnction type.
           If the data should be dynamically allocated by the configuration
           software it should be a pointer to the pointer that should
           be set. Allocated data is initialized with '0's.
        5) A function pointer. Must not be NULL except for "FUnction" type
           and is optional (may be NULL) for data type entries.
           For the "Function" type, the data (normally a character string)
           is passed as the only argument. For data type entries,
           the associated functions are called with an extended
           calling syntax.
        6) A pointer to a character string with the default initialization
           values or NULL.
        7) A pointer to a character string with a lower bound value or NULL.
        8) A pointer to a character string with an upper bound value or NULL.
        9) An integer where any of the following flags may be combined
           by a bitwise OR '|':
              CFG_REQUIRE_DATA
              CFG_REQUIRE_ALL_DATA
              CFG_REJECT_MODIFICATION
       10) Reserved. In multi_threaded mode, use
              CFG_MUTEX(&some_pthread_mutex)
           if the associated function is not fully reentrant or
           if a set of functions should only be called one at a time.
       11) Reserved. Do not modify. Is 1 if reconfigured.
```

Components not specified are automatically initialized to NULL or 0.

The reason why build_config may be called several times (with different configuration items each time) is that this way the configuration items for each more or less independent part of a program may be defined separately and there is no need for global data sharing. You only need to call a 'configuration definition function' for each part which has its items defined and only calls build_config().

Once the whole configuration items from all parts have been passed to build_config(), a single call to init_config() is required to make the configuration effective. init_config() first sets those initial values declared in the items (if any) and then tries to get external data line by line from a function passed to init_config(), unless a NULL pointer is passed instead of a function pointer. This user-defined function (declared 'char ∗user_function(void);') should return the address of the first character of each line read from a configuration file, the command line, or anywhere else, until the end of input which the function must indicate by returning a NULL pointer. Input lines can be of any length up to 10240 bytes and may include a linefeed character as read by fgets(). Note that there used to be a problem with semicolons in comments, which should be fixed now - but beware of possible side-effects.

Later, configuration data can be changed by calling reconfig() with a line of input passed as argument. Configuration data marked as 'not to be modified' will not be changed. If a configuration item is of 'function' type that function will be called with the remaining line (after extracting the item name and processing special characters) passed as argument.

7.25.2 Function Documentation

7.25.2.1 build_config()

```
int build_config (
            CONFIG_ITEM * items,
            const char * section )
```

Build up the configuration by adding another section of configuration definitions.

Parameters

items	Vector of configuration items, which is terminated by a NULL_CONFIG_ITEM
section	Name of this configuration section.

Returns

0 (O.k.), -1 (memory allocation failed), -2 (other error)

7.25.2.2 find_config_item()

```
CONFIG_ITEM* find_config_item (
            const char * name )
```

Find a configuration item by its name (mainly for internal usage).

Parameters

name	Item name or block:name

Returns

Pointer to (first) configuration item found or NULL.

7.25.2.3 get_config_current()

```
const char* get_config_current (
            CONFIG_ITEM * item,
            char * buf,
            size_t lbuf )
```

Retrieve current settings of a single configuration item as a text string.

The formatted current configuration (same as with 'SHOW' or 'LIST') is returned to user code in a user-provided buffer. If the buffer is too short, no or partial results may be returned.

Parameters

item	Pointer to selected configuration item structure
buf	Buffer to be filled with formatted configuration content
lbuf	Size of provided buffer (including any trailing '\0' character).

Returns

Current configuration in text representation, as far as fitting into buffer.

References ConfigItemStruct::data, ConfigItemStruct::internal, ConfigIntern::itype, and ConfigItemStruct::size.

7.25.2.4 get_config_filename()

```
char* get_config_filename (
            void )
```

Return the current value of the configuration file name.

Parameters

–	(none)

Returns

pointer to static file name string

7.25.2.5 get_config_preprocessor()

```
char* get_config_preprocessor (
            void )
```

Return the current value of the configuration preprocessor.

Parameters

–	(none)

Returns

pointer to static command string

7.25.2.6 init_config()

```
int init_config (
            char *(*)(void) fptr )
```

Initialize the configuration after all build_config() calls.

Initialize the configuration after all sections have been supplied via build_config(). A function may be specified for reading external configuration data after the internal specifications have been processed. This function may be called only once.

Parameters

fptr	Pointer to function that returns a string pointer as long as external configuration data is available, and NULL when no more data is available. fptr may be NULL if no such function should be called.

Returns

0 (O.k.), -1 (called a second time or invalid configuration data)

7.25.2.7 read_config_lines()

```
char* read_config_lines (
            void )
```

Read configuration data from a file and return it line by line to the calling function (one line per call).

A NULL pointer is returned on end-of-file. This function is intended to be used as the usual 'fptr' argument for init_config().

Parameters

–	(none)

Returns

Pointer to character string or NULL.

7.25.2.8 read_config_status()

```
int read_config_status (
            void  )
```

Return the status of reading a configuration file with read_config_lines() in a preceding call to init_config().

Parameters

–	(none)

Returns

0 (o.k.), -1 (no config file set), -2 (config file open failed), -3 (preprocessing failed), -4 (read error).

7.25.2.9 reconfig()

```
int reconfig (
            char * text )
```

Modify the configuration after init_config() has been called.

Parameters

text	String consisting of configuration keyword (separated by a blank or '=' from the rest) and the corresponding data.

Returns

0 (O.k.), -1 (invalid or undefined configuration keyword or error in the data)

7.25.2.10 reload_config()

```
int reload_config (
            char *(*)(void) fptr )
```

Reload some configuration using the file name/preprocessor as set up for init_config() or with different file etc.

Parameters

fptr	Pointer to function that returns a string pointer as long as external configuration data is available, and NULL when no more data is available.

Returns

0 (O.k.), -1 (invalid configuration data)

7.25.2.11 set_config_filename()

```
void set_config_filename (
             const char * fname )
```

Set the name of the configuration file to be read by the function read_config_lines().

Parameters

fname	Name of file to be used.

Returns

(none)

7.25.2.12 set_config_history()

```
int set_config_history (
             PFITI fptr )
```

Set a function for recording the history of the configuration settings.

Parameters

fptr	– Pointer to function of type 'int fptr(char ∗text,int flag)' where 'text' is the configuration line and flag is 0 for configuration file processing and 1 for latre reconfiguration.

Returns

0

7.25.2.13 set_config_preprocessor()

```
void set_config_preprocessor (
             char * preproc )
```

Set the command name and options of a preprocessor for configuration files to be read by function read_config_lines().

The input and output file names will be appended to the command string set by this function.

Parameters

preproc	Command string

Returns

(none)

7.25.2.14 set_config_stack()

```
void set_config_stack (
             char ** stack )
```

Set a list of configuration lines to be processed before any lines from a file are read by read_config_lines().

Parameters

stack	Pointer to NULL terminated vector of strings.

Returns

(none)

7.25.3 Variable Documentation

7.25.3.1 config_defaults

```
struct config_specific_data config_defaults  [static]
```

Initial value:

```
=
{
   "_internal_"
}
```

7.25.3.2 default_config

```
CONFIG_ITEM default_config[ ]  [static]
```

Initial value:

```
=
{

   { "SHOW",    "FUN", -1, NULL, f_show_config,   NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "LOCK",    "FUN", -1, NULL, f_lock_config,   NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "UNLOCK",  "FUN", -1, NULL, f_unlock_config, NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "LIMITS",  "FUN", -1, NULL, f_limit_config,  NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "STATUS",  "FUN", -1, NULL, f_status_config, NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "LIST",    "FUN", -1, NULL, f_list_config,   NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "INITLIST","FUN",-1, NULL, f_initlist_config, NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "TYPELIST","FUN",-1, NULL, f_typelist_config, NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "GET",     "FUN", -1, NULL, f_get_config,    NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "ECHO",    "FUN", -1, NULL, f_echo,          NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "WARNING","FUN", -1, NULL, f_warning,        NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { "ERROR",   "FUN", -1, NULL, f_error,         NULL, NULL, NULL, 0, NULL, CFG_MUTEX(mlock_hconfig) },
   { NULL_CONFIG_ITEM }
}
```

Internal functions of the hconfig package.

7.25.3.3 first_config_block

```
CONFIG_BLOCK first_config_block  [static]
```

Initial value:

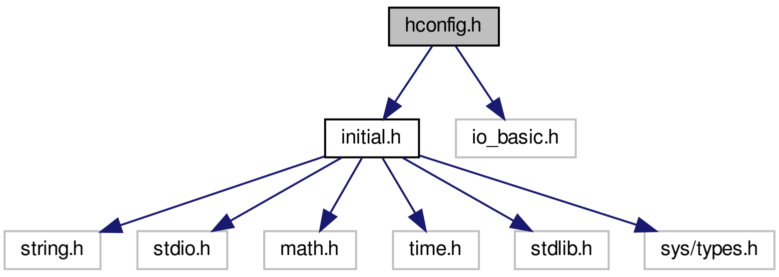
```
=
   { "_internal_", default_config, (CONFIG_BLOCK *) NULL, 0 }
```

7.26 hconfig.h File Reference

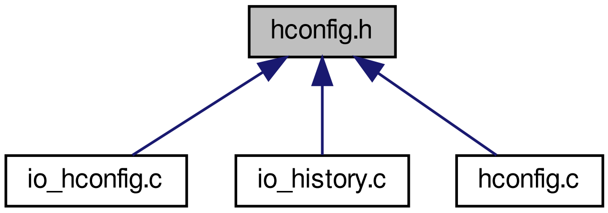
Declare hconfig structures and functions.

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

Include dependency graph for hconfig.h:



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



Data Structures

- union ConfigDataPointer

  This union of pointers allows convenient access of various types of data.
- union ConfigBoundary

  Configuration value may have optional lower and/or upper bounds.
- struct ConfigValues

  Configuration values and supporting data passed to user functions.
- struct Config_Binary_Item_Interface

  Interface definitions for binary-only items.
- struct ConfigIntern

  Configuration elements used only internally.
- struct ConfigItemStruct

  Configuration as used in definitions of configuration blocks.

Macros

- #define **NO_INITIAL_MACROS** 1
- #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 **CONST** const
- #define **IO_TYPE_HCONFIG_ENVELOPE** 900
- #define **IO_TYPE_HCONFIG_NAME** 901
- #define **IO_TYPE_HCONFIG_TEXT** 902
- #define **IO_TYPE_HCONFIG_INDEX** 903
- #define **IO_TYPE_HCONFIG_NUMBERS** 904
- #define **CFG_REQUIRE_DATA** 1
- #define **CFG_REQUIRE_ALL_DATA** 2
- #define **CFG_REJECT_MODIFICATION** 4
- #define **CFG_HARD_BOUND** 8
- #define **CFG_STRICT_BOUND** 16
- #define **CFG_INITIALIZED** 32
- #define **CFG_ALL_INITIALIZED** 64
- #define **CFG_NOT_INITIAL** 128
- #define **NULL_CONFIG_ITEM** (char ∗) NULL, (char ∗) NULL, 0, NULL, NULL, (char ∗) NULL, (char ∗) NULL, (char ∗) NULL, 0, NULL, NULL, NULL, {0}
- #define CFG_MUTEX(mutex) (NULL)

  Mutexes are only inserted when pthreads are used.

Typedefs

- typedef unsigned char **bool**
- typedef void ∗(∗ **PFVP**) (char ∗, char ∗, int)
- typedef int(∗ **PFISI**) (char ∗, int)
- typedef int(∗ **PFITI**) (const char ∗, int)
- typedef int(∗ **PFISS**) (char ∗, char ∗)
- typedef struct ConfigValues **CONFIG_VALUES**
- typedef int(∗ **PFIX**) (const char ∗name, CONFIG_VALUES ∗val)
- typedef struct ConfigItemStruct **CONFIG_ITEM**

Functions

- int build_config (CONFIG_ITEM ∗items, const char ∗section)

  Build up the configuration by adding another section of configuration definitions.
- int init_config (char ∗(∗fptr)(void))

  Initialize the configuration after all build_config() calls.
- void unhook_internal (void)

  Disable access to internal functions via configuration.
- void rehook_internal (void)

  Enable access again to internal functions via configuration.
- int reload_config (char ∗(∗fptr)(void))

  Reload some configuration using the file name/preprocessor as set up for init_config() or with different file etc.
- void ∗ **config_alloc_data** (char ∗name, char ∗type, int size)
- int reconfig (char ∗text)

  Modify the configuration after init_config() has been called.
- int **verify_config_section** (char ∗section)
- int set_config_history (PFITI fptr)

  Set a function for recording the history of the configuration settings.
- void set_config_filename (const char ∗fname)

  Set the name of the configuration file to be read by the function read_config_lines().
- char ∗ get_config_filename (void)

  Return the current value of the configuration file name.
- void set_config_preprocessor (char ∗preproc)

  Set the command name and options of a preprocessor for configuration files to be read by function read_config_lines().
- char ∗ get_config_preprocessor (void)

  Return the current value of the configuration preprocessor.
- void set_config_stack (char ∗∗stack)

  Set a list of configuration lines to be processed before any lines from a file are read by read_config_lines().
- char ∗ read_config_lines (void)

  Read configuration data from a file and return it line by line to the calling function (one line per call).
- int read_config_status (void)

  Return the status of reading a configuration file with read_config_lines() in a preceding call to init_config().
- CONFIG_ITEM ∗ find_config_item (const char ∗name)

  Find a configuration item by its name (mainly for internal usage).
- int define_config_binary_interface (int item_type, size_t elem_size, void ∗(∗new_func)(int nelem, int item↩_type), int(∗delete_func)(void ∗ptr, int nelem, int item_type), int(∗read_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type), int(∗write_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type), int(∗readtext_↩func)(void ∗bin_item, char ∗text, int item_type), int(∗list_func)(void ∗bin_item, int item_type), int(∗copy_↩func)(void ∗bin_item_to, void ∗bin_item_from, int io_type))

  Define a binary interface for an I/O type.

- struct Config_Binary_Item_Interface ∗ find_config_binary_interface (int item_type)

  Find the matching binary interface for given item type.
- int **reconfig_binary** (char ∗buffer, size_t buflen)
- int config_binary_read_text (IO_BUFFER ∗iobuf, char ∗name, int maxlen)

  Get a hconfig name or text item from an I/O buffer.
- const char ∗ get_config_current (CONFIG_ITEM ∗item, char ∗buf, size_t lbuf)

  Retrieve current settings of a single configuration item as a text string.
- int **is_signed_number** (const char ∗text)
- int **is_unsigned_number** (const char ∗text)
- int **is_hex_number** (const char ∗text)
- int **is_bin_number** (const char ∗text)
- int **is_real_number** (const char ∗text)
- int **is_boolean** (const char ∗text)
- unsigned long **decode_bin_number** (const char ∗text)
- 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 > delimeted 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 config_binary_read_index (IO_BUFFER ∗iobuf, int ∗nidx, int ∗idx_low, int ∗idx_high, int max_idx)

  Get a list of index ranges for binary hconfig data following.
- int config_binary_write_name (IO_BUFFER ∗iobuf, char ∗name)

  Write the name of a hconfig item for which binary data should follow.
- int config_binary_write_text (IO_BUFFER ∗iobuf, char ∗text)

  Write 'binary' hconfig data as text (for 'string' or 'function' types).
- int config_binary_text_length (IO_BUFFER ∗iobuf)

  If the next item is of the text type, get the length of the text.
- int config_binary_read_name (IO_BUFFER ∗iobuf, char ∗name, int maxlen)

  Is the same as config_binary_read_text().
- int config_binary_write_index (IO_BUFFER ∗iobuf, int nidx, int ∗idx_low, int ∗idx_high)

  Put a list of index ranges for binary hconfig data following.
- int config_binary_envelope_begin (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Begin with the envelope for a binary configuration item.
- int config_binary_envelope_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Close the envelope for a binary configuration item.
- int config_binary_inquire_numbers (IO_BUFFER ∗iobuf, int ∗ntype, int ∗nsize, int32_t ∗num, int ∗nopt)

  Tell me what kind of binary numbers follow in the next I/O item.
- int config_binary_read_numbers (IO_BUFFER ∗iobuf, void ∗data, size_t max_size)

  Get the binary numbers from the next I/O item.
- int config_binary_convert_data (void ∗out, int out_type, int out_size, void ∗in, int in_type, int in_size)

  Concert binary numbers of one type to numbers of another type.

7.26.1 Detailed Description

Declare hconfig structures and functions.

Author

Konrad Bernloehr

Date

1993 to 2023

7.26.2 Macro Definition Documentation

7.26.2.1 CFG_MUTEX

```
#define CFG_MUTEX(
              mutex ) (NULL)
```

Mutexes are only inserted when pthreads are used.

In the multi-threaded variant: the address of the given mutex. In the single-threaded variant: a null pointer.

7.26.3 Function Documentation

7.26.3.1 abbrev()

```
int abbrev (
             CONST char * s,
             CONST char * t )
```

Compare strings s and 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

s	The string to be checked.
t	The test string with minimum part in upper case.

Returns

1 if s is an abbreviation of t, 0 if not.

7.26.3.2 build_config()

```
int build_config (
             CONFIG_ITEM * items,
             const char * section )
```

Build up the configuration by adding another section of configuration definitions.

Parameters

items	Vector of configuration items, which is terminated by a NULL_CONFIG_ITEM
section	Name of this configuration section.

Returns

0 (O.k.), -1 (memory allocation failed), -2 (other error)

7.26.3.3 config_binary_convert_data()

```
int config_binary_convert_data (
            void * out,
            int out_type,
            int out_size,
            void * in,
            int in_type,
            int in_size )
```

Concert binary numbers of one type to numbers of another type.

Supported types are signed integers of various lengths, unsigned integers of various lengths, float and double. The signed and unsigned integers can be 1, 2, 4 or perhaps 8 bytes long. Float should be 4 bytes long, double 8 bytes.

7.26.3.4 config_binary_read_text()

```
int config_binary_read_text (
            IO_BUFFER * iobuf,
            char * name,
            int maxlen )
```

Get a hconfig name or text item from an I/O buffer.

Both the IO_TYPE_HCONFIG_NAME and IO_TYPE_HCONFIG_TEXT eventio item types are simple text strings enclosed in an I/O item. Because either of them can appear at the beginning of binary configuration data (with different interpretations) they are distinguished by different item type numbers. Otherwise they are the same.

Referenced by config_binary_read_name().

7.26.3.5 config_binary_text_length()

```
int config_binary_text_length (
            IO_BUFFER * iobuf )
```

If the next item is of the text type, get the length of the text.

This allows finding out the length of the text first, allocating enough memory to read it and then start reading the text.

Returns

The length of the string not including the trailing '\0' which has to be appended.

7.26.3.6 config_binary_write_name()

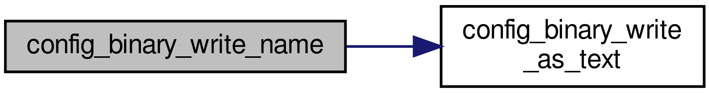
```
int config_binary_write_name (
            IO_BUFFER * iobuf,
            char * name )
```

Write the name of a hconfig item for which binary data should follow.

Calls config_binary_write_as_text().

References config_binary_write_as_text().

Here is the call graph for this function:



7.26.3.7 config_binary_write_text()

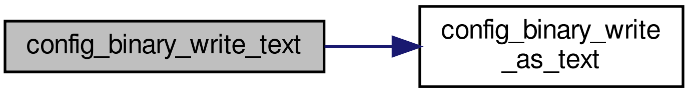
```
int config_binary_write_text (
            IO_BUFFER * iobuf,
            char * text )
```

Write 'binary' hconfig data as text (for 'string' or 'function' types).

Calls config_binary_write_as_text().

References config_binary_write_as_text().

Here is the call graph for this function:



7.26.3.8 find_config_item()

```
CONFIG_ITEM* find_config_item (
            const char * name )
```

Find a configuration item by its name (mainly for internal usage).

Parameters

name	Item name or block:name

Returns

Pointer to (first) configuration item found or NULL.

7.26.3.9 get_config_current()

```
const char* get_config_current (
            CONFIG_ITEM * item,
            char * buf,
            size_t lbuf )
```

Retrieve current settings of a single configuration item as a text string.

The formatted current configuration (same as with 'SHOW' or 'LIST') is returned to user code in a user-provided buffer. If the buffer is too short, no or partial results may be returned.

Parameters

item	Pointer to selected configuration item structure
buf	Buffer to be filled with formatted configuration content
lbuf	Size of provided buffer (including any trailing '\0' character).

Returns

Current configuration in text representation, as far as fitting into buffer.

References ConfigItemStruct::data, ConfigItemStruct::internal, ConfigIntern::itype, and ConfigItemStruct::size.

7.26.3.10 get_config_filename()

```
char* get_config_filename (
            void )
```

Return the current value of the configuration file name.

Parameters

–	(none)

Returns

pointer to static file name string

7.26.3.11 get_config_preprocessor()

```
char* get_config_preprocessor (
            void  )
```

Return the current value of the configuration preprocessor.

Parameters

–	(none)

Returns

pointer to static command string

7.26.3.12 getword()

```
int getword (
            CONST char * s,
            int * spos,
            char * word,
            int maxlen,
            char blank,
            char endchar )
```

Copies a blank or '\0' or $<$ endchar $>$ delimeted 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.

The word must have a length less than or equal to maxlen.

Parameters

s	string with any number of words.
spos	position in the string where we start and end.
word	the extracted word.
maxlen	the maximum allowed length of word.
blank	has the same effect as ' ', i.e. end-of-word.
endchar	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

Referenced by push_config_history(), and user_set_tel_type_param_by_str().

7.26.3.13 init_config()

```
int init_config (
            char *(*)(void) fptr )
```

Initialize the configuration after all build_config() calls.

Initialize the configuration after all sections have been supplied via build_config(). A function may be specified for reading external configuration data after the internal specifications have been processed. This function may be called only once.

Parameters

fptr	Pointer to function that returns a string pointer as long as external configuration data is available, and NULL when no more data is available. fptr may be NULL if no such function should be called.

Returns

0 (O.k.), -1 (called a second time or invalid configuration data)

7.26.3.14 read_config_lines()

```
char* read_config_lines (
            void )
```

Read configuration data from a file and return it line by line to the calling function (one line per call).

A NULL pointer is returned on end-of-file. This function is intended to be used as the usual 'fptr' argument for init_config().

Parameters

–	(none)

Returns

Pointer to character string or NULL.

7.26.3.15 read_config_status()

```
int read_config_status (
            void  )
```

Return the status of reading a configuration file with read_config_lines() in a preceding call to init_config().

Parameters

–	(none)

Returns

0 (o.k.), -1 (no config file set), -2 (config file open failed), -3 (preprocessing failed), -4 (read error).

7.26.3.16 reconfig()

```
int reconfig (
            char * text )
```

Modify the configuration after init_config() has been called.

Parameters

text	String consisting of configuration keyword (separated by a blank or '=' from the rest) and the corresponding data.

Returns

0 (O.k.), -1 (invalid or undefined configuration keyword or error in the data)

7.26.3.17 reload_config()

```
int reload_config (
            char *(*)(void) fptr )
```

Reload some configuration using the file name/preprocessor as set up for init_config() or with different file etc.

Parameters

fptr	Pointer to function that returns a string pointer as long as external configuration data is available, and NULL when no more data is available.

Returns

0 (O.k.), -1 (invalid configuration data)

7.26.3.18 set_config_filename()

```
void set_config_filename (
             const char * fname )
```

Set the name of the configuration file to be read by the function read_config_lines().

Parameters

fname	Name of file to be used.

Returns

(none)

7.26.3.19 set_config_history()

```
int set_config_history (
             PFITI fptr )
```

Set a function for recording the history of the configuration settings.

Parameters

fptr	– Pointer to function of type 'int fptr(char ∗text,int flag)' where 'text' is the configuration line and flag is 0 for configuration file processing and 1 for latre reconfiguration.

Returns

0

7.26.3.20 set_config_preprocessor()

```
void set_config_preprocessor (
             char * preproc )
```

Set the command name and options of a preprocessor for configuration files to be read by function read_config_lines().

The input and output file names will be appended to the command string set by this function.

Parameters

preproc	Command string

Returns

(none)

7.26.3.21 set_config_stack()

```
void set_config_stack (
            char ** stack )
```

Set a list of configuration lines to be processed before any lines from a file are read by read_config_lines().

Parameters

stack	Pointer to NULL terminated vector of strings.

Returns

(none)

7.27 hessio_doc.h File Reference

Add an introduction to doxygen-generated documentation.

7.27.1 Detailed Description

Add an introduction to doxygen-generated documentation.

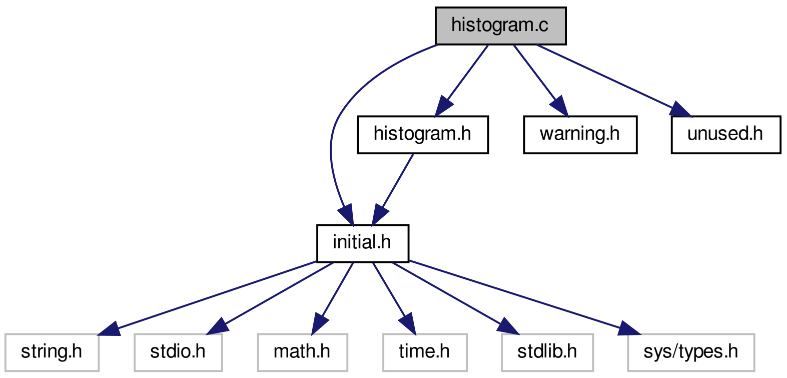
This file is not included during compilation.

7.28 histogram.c File Reference

Manage, fill, and display one- and two-dimensional histograms.

```
#include "initial.h"
#include "histogram.h"
#include "warning.h"
```

`#include "unused.h"`
Include dependency graph for histogram.c:



Macros

- #define **_HLOCK_**
- #define **_HUNLOCK_**
- #define **_WAIT_IF_BUSY_**(histo)
- #define **_CLEAR_BUSY_**(histo)
- #define **HistOutput**(a)

Functions

- static void initialize_histogram (HISTOGRAM ∗histo)

  For internal purpose only.
- static HISTOGRAM ∗ aux_alloc_histogram (int ncounts, const char ∗type)

  For internal purpose only.
- static void free_histo_contents (HISTOGRAM ∗histo)

  Free the contents (data pointers) of a histogram to be released or removed.
- static void display_2d_histogram (HISTOGRAM ∗histo)

  Display contents of a 2D histogram.
- void **histogram_lock** (_unused_ HISTOGRAM ∗histo)
- void **histogram_unlock** (_unused_ HISTOGRAM ∗histo)
- HISTOGRAM ∗ get_first_histogram ()

  Get a pointer to the first histogram.
- void sort_histograms ()

  Sort histograms in linked list by idents.
- void set_first_histogram (HISTOGRAM ∗new_first_histogram)

  Set a new histogram as the first element (context switching).
- HISTOGRAM ∗ get_histogram_by_ident (long ident)

  Get a histogram with the given ID.
- void list_histograms (long ident)

  List all available histograms using the 'Output()' function.

- HISTOGRAM ∗ book_histogram (long id, const char ∗title, const char ∗type, int dimension, double ∗low, double ∗high, int ∗nbins)

    General histogram booking function, assigning ID and title.
- HISTOGRAM ∗ book_1d_histogram (long id, const char ∗title, const char ∗type, double low, double high, int nbins)

    Simplified histogram booking function for one-dimensional histograms, assigning ID and title.
- HISTOGRAM ∗ book_int_histogram (long id, const char ∗title, int dimension, long ∗low, long ∗high, int ∗nbins)

    Book and integer-type histogram (content incremented by one per entry).
- HISTOGRAM ∗ allocate_histogram (const char ∗type, int dimension, double ∗low, double ∗high, int ∗nbins)

    Allocate any histogram without ID and title.
- HISTOGRAM ∗ alloc_int_histogram (long low, long high, int nbins)

    Allocate memory for a 1-D 'int' histogram and initialize it.
- HISTOGRAM ∗ alloc_real_histogram (double low, double high, int nbins)

    Allocate memory for a 1-D 'real' histogram and initialize it.
- HISTOGRAM ∗ alloc_2d_int_histogram (long xlow, long xhigh, int nxbins, long ylow, long yhigh, int nybins)

    Allocate memory for a 2-D 'int' histogram and initialize it.
- HISTOGRAM ∗ alloc_2d_real_histogram (double xlow, double xhigh, int nxbins, double ylow, double yhigh, int nybins)

    Allocate memory for a 2-D 'int' histogram and initialize it.
- void describe_histogram (HISTOGRAM ∗histo, const char ∗title, long ident)

    Add a describing title to a histogram previously allocated.
- void clear_histogram (HISTOGRAM ∗histo)

    Initialize an existing histogram.
- void free_histogram (HISTOGRAM ∗histo)

    Free a histogram completely (both data and control structure).
- void free_all_histograms ()

    Deletes all histograms which are included in the linked list of histograms.
- void unlink_histogram (HISTOGRAM ∗histo)

    Remove a histogram from the list without destroying it.
- int fill_int_histogram (HISTOGRAM ∗histo, long value)

    Increment a bin of a 1-D 'int' histogram by one.
- int fill_real_histogram (HISTOGRAM ∗histo, double value)

    Increment a bin of a 1-D 'real' histogram by one.
- int fill_weighted_histogram (HISTOGRAM ∗histo, double value, double weight)

    Add an entry to a weighted 1-D histogram.
- int fill_2d_int_histogram (HISTOGRAM ∗histo, long xvalue, long yvalue)

    Increment a bin of a 2-D 'int' histogram by one.
- int fill_2d_real_histogram (HISTOGRAM ∗histo, double xvalue, double yvalue)

    Increment a bin of a 2-D 'real' histogram by one.
- int fill_2d_weighted_histogram (HISTOGRAM ∗histo, double xvalue, double yvalue, double weight)

    Add an entry to a weighted 2-D histogram.
- int fill_histogram (HISTOGRAM ∗histo, double xvalue, double yvalue, double weight)

    Fill any type of 1-D or 2-D histogram known by its pointer.
- int fill_histogram_by_ident (long id, double xvalue, double yvalue, double weight)

    Fill any type of 1-D or 2-D histogram known by its ID number.
- int histogram_matching (HISTOGRAM ∗histo1, HISTOGRAM ∗histo2)

    Check if two histograms have exactly matching definitions (same type, dimension, size, ranges).
- HISTOGRAM ∗ add_histogram (HISTOGRAM ∗histo1, HISTOGRAM ∗histo2)

    Add a second histogram to a first one.
- int stat_histogram (HISTOGRAM ∗histo, struct histstat ∗stbuf)

    Statistical analysis of a histogram.

- double locate_histogram_fraction (HISTOGRAM ∗histo, double fraction)

  Locate point of arbitrary fraction of entries (quantile).
- int fast_stat_histogram (HISTOGRAM ∗histo, struct histstat ∗stbuf)

  Fast and basic histogram statistics.
- void print_histogram_scaled (HISTOGRAM ∗histo, double fact)

  Print scaled contents of a histogram on the terminal.
- void print_histogram (HISTOGRAM ∗histo)

  Print contents of a histogram on the terminal.
- void display_histogram (HISTOGRAM ∗histo)

  Display contents of a histogram on the terminal.
- void display_all_histograms ()

  Display all histograms in list of histograms.
- int histogram_to_lookup (HISTOGRAM ∗histo, HISTOGRAM ∗lookup)

  Convert a histogram to a lookup table by integrating the histogram.
- long lookup_int (HISTOGRAM ∗lookup, long value, long factor)

  Look up a table created from an integer histogram.
- double lookup_real (HISTOGRAM ∗lookup, double value, double factor)

  Look up a table created from an 'real' histogram.
- int histogram_hashing (int tabsize)

  Turn hashing of histograms (using their ident as key) on or off.

Variables

- static HISTOGRAM ∗ **first_histogram** = (HISTOGRAM ∗) NULL
- static HISTOGRAM ∗ **last_histogram** = (HISTOGRAM ∗) NULL
- FILE ∗ **histogram_file**
- static HISTOGRAM ∗∗ **hash_table**
- static long **hash_size** = 0
- static CONST_QUAL short **primetab** [ ]
- static CONST_QUAL int **zero** = 0

7.28.1 Detailed Description

Manage, fill, and display one- and two-dimensional histograms.

Eventio routines for these types of histograms are available in io_histogram.c. Conversion to HBOOK format is available through the `hdata2hbook` (was `cvt2`) program. Conversion to ROOT format is available through the `hdata2root` (was `cvt3`) program.

Note: multi-threading safety of functions provided in this file has not been tested extensively. Threads must not delete histograms shared with other threads when referenced by pointers.

Author

Konrad Bernloehr

Date

1991 to 2023

7.28.2 Macro Definition Documentation

7.28.2.1 HistOutput

```
#define HistOutput(
               a )
```

Value:

```
       do { if ( histogram_file == (FILE *) NULL ) \
       Output(a); \
    else \
       fputs(a,histogram_file); } while(zero)
```

7.28.3 Function Documentation

7.28.3.1 add_histogram()

```
HISTOGRAM* add_histogram (
              HISTOGRAM * histo1,
              HISTOGRAM * histo2 )
```

Add a second histogram to a first one.

The histograms must exactly match in their definitions. The first histogram will be modified, the second is unchanged.

Parameters

histo1	pointer to first histogram
histo2	pointer to second histogram

Returns

NULL pointer indicates failure.

7.28.3.2 alloc_2d_int_histogram()

```
HISTOGRAM* alloc_2d_int_histogram (
              long xlow,
              long xhigh,
              int nxbins,
              long ylow,
              long yhigh,
              int nybins )
```

Allocate memory for a 2-D 'int' histogram and initialize it.

Resulting histogram has integer range limits and integer contents (incremented by one per entry).

Parameters

xlow	lower limit of values in X to be covered by histogram
xhigh	upper limit ...
nxbins	the number of bins to be allocated in X
ylow	lower limit of values in Y to be covered by histogram
yhigh	upper limit ...
nybins	the number of bins to be allocated in Y

Returns

pointer to allocated histogram or NULL

References aux_alloc_histogram().

Here is the call graph for this function:

alloc_2d_int_histogram
aux_alloc_histogram

7.28.3.3 alloc_2d_real_histogram()

```
HISTOGRAM* alloc_2d_real_histogram (
            double xlow,
            double xhigh,
            int nxbins,
            double ylow,
            double yhigh,
            int nybins )
```

Allocate memory for a 2-D 'int' histogram and initialize it.

Resulting histogram has floating point range limits and integer contents (incremented by one per entry).

Parameters

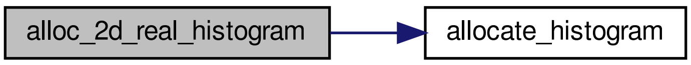
xlow	lower limit of values in X to be covered by histogram
xhigh	upper limit ...
nxbins	the number of bins to be allocated in X
ylow	lower limit of values in Y to be covered by histogram
yhigh	upper limit ...
nybins	the number of bins to be allocated in Y

Returns

pointer to allocated histogram or NULL

References allocate_histogram().

Here is the call graph for this function:



7.28.3.4 alloc_int_histogram()

```
HISTOGRAM* alloc_int_histogram (
              long low,
              long high,
              int nbins )
```

Allocate memory for a 1-D 'int' histogram and initialize it.

Resulting histogram has integer range limits and integer contents (incremented by one per entry).

Parameters

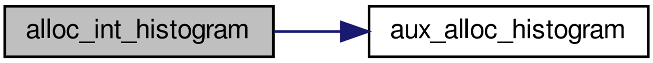
low	lower limit of values to be covered by histogram
high	upper limit ...
nbins	the number of bins to be allocated

Returns

pointer to allocated histogram or NULL

References aux_alloc_histogram().

Here is the call graph for this function:



7.28.3.5 alloc_real_histogram()

```
HISTOGRAM* alloc_real_histogram (
             double low,
             double high,
             int nbins )
```

Allocate memory for a 1-D 'real' histogram and initialize it.

Resulting histogram has floating point range limits and integer contents (incremented by one per entry).

Parameters

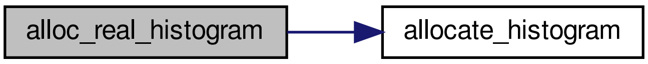
low	lower limit of values to be covered by histogram
high	upper limit ...
nbins	the number of bins to be allocated

Returns

pointer to allocated histogram or NULL

References allocate_histogram().

Here is the call graph for this function:



7.28.3.6 allocate_histogram()

```
HISTOGRAM* allocate_histogram (
             const char * type,
             int dimension,
             double * low,
             double * high,
             int * nbins )
```

Allocate any histogram without ID and title.

Allocate a histogram of 1 or 2 dimensions, 'I', 'R', 'F' or 'D' type, without assigning an ID number and title string to it. To avoid the (long) <--> (double) typecasts, the direct calls to alloc_int_histogram() and alloc_2d_int_histogram() are recommended for integer-limits histograms (type 'I').

Parameters

type	"I" (int, no weights), "R" (real, no weights), "F" (float, with weights), "D" (double, w.w.)
dimension	1 or 2 for 1-D or 2-D histogram
low	Pointer to lower limits (x or x,y for 1-D or 2-D)
high	Pointer to upper limits
nbins	Pointer to no. of bins per dimension (nx or nx, ny)

Returns

Pointer to new histogram or NULL

Referenced by alloc_2d_real_histogram(), alloc_real_histogram(), book_1d_histogram(), and book_histogram().

7.28.3.7 book_1d_histogram()

```
HISTOGRAM* book_1d_histogram (
             long id,
             const char * title,
             const char * type,
             double low,
             double high,
             int nbins )
```

Simplified histogram booking function for one-dimensional histograms, assigning ID and title.

Book a histogram of one dimension, 'I', 'R', 'F', or 'D' type. The histogram is allocated (if possible) and the supplied ID number and title string are assigned.

Parameters

id	ID number
title	Histogram title string
type	"I" (int, no weights), "R" (real, no weights), "F" (float, with weights), "D" (double, w.w.)
low	Lower limit (x)
high	Upper limit (x)
nbins	No. of bins (nx)

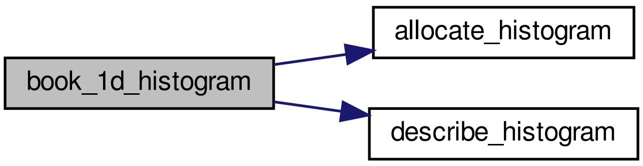
Returns

Pointer to new histogram or NULL

References allocate_histogram(), and describe_histogram().

Referenced by project_histogram().

Here is the call graph for this function:



7.28.3.8 book_histogram()

```
HISTOGRAM* book_histogram (
             long id,
             const char * title,
             const char * type,
             int dimension,
             double * low,
             double * high,
             int * nbins )
```

General histogram booking function, assigning ID and title.

Book a histogram of 1 or 2 dimensions, 'I', 'R', 'F', or 'D' type. The histogram is allocated (if possible) and the supplied ID number and title string are assigned.

Parameters

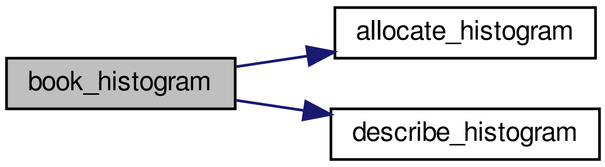
id	ID number
title	Histogram title string
type	"I" (int, no weights), "R" (real, no weights), "F" (float, with weights), "D" (double, w.w.)
dimension	1 or 2 for 1-D or 2-D histogram
low	Pointer to lower limits (x or x,y for 1-D or 2-D)
high	Pointer to upper limits
nbins	Pointer to no. of bins per dimension (nx or nx, ny)

Returns

Pointer to new histogram or NULL

References allocate_histogram(), and describe_histogram().

Here is the call graph for this function:



7.28.3.9 book_int_histogram()

```
HISTOGRAM* book_int_histogram (
             long id,
             const char * title,
             int dimension,
             long * low,
             long * high,
             int * nbins )
```

Book and integer-type histogram (content incremented by one per entry).

Like book_histogram() but for 'I' type histograms only (1-D or 2-D)

Parameters

id	ID number
title	Histogram title string
dimension	1 or 2 for 1-D or 2-D histogram
low	Pointer to lower limits (x or x,y for 1-D or 2-D)
high	Pointer to upper limits
nbins	Pointer to no. of bins per dimension (nx or nx, ny)

Returns

Pointer to new histogram or NULL

7.28.3.10 clear_histogram()

```
void clear_histogram (
            HISTOGRAM * histo )
```

Initialize an existing histogram.

Parameters

histo	– pointer to histogram

Returns

(none)

Referenced by write_dst_histos().

7.28.3.11 describe_histogram()

```
void describe_histogram (
            HISTOGRAM * histo,
            const char * title,
            long ident )
```

Add a describing title to a histogram previously allocated.

Parameters

histo	Histogram to which the title should be added
title	The title string. This is ignored if the histogram already has a title.
ident	Identification number, must be unique (or 0) if any I/O is intended, because read_histogram() deletes a pre-existing histogram with the same ID.

Returns

none

Referenced by book_1d_histogram(), and book_histogram().

7.28.3.12 display_2d_histogram()

```
static void display_2d_histogram (
            HISTOGRAM * histo )  [static]
```

Display contents of a 2D histogram.

Called by display_histogram().

The histogram has already been checked by display_histogram() and its title has been printed.

Parameters

histo	– Pointer to histogram

Returns

(none)

References histogram::counts, Histogram_Extension::ddata, histogram::extension, Histogram_Extension::fdata, histogram::nbins, histogram::nbins_2d, and histogram::type.

7.28.3.13 display_all_histograms()

```
void display_all_histograms (
            void  )
```

Display all histograms in list of histograms.

Arguments: none

Return value: none

7.28.3.14 display_histogram()

```
void display_histogram (
            HISTOGRAM * histo )
```

Display contents of a histogram on the terminal.

This is a simple 'HPRINT' type display on one screen.

Parameters

histo	Pointer to histogram

Returns

(none)

References histogram::counts, histogram::extension, histogram::nbins, histogram::tentries, and histogram::type.

7.28.3.15 fast_stat_histogram()

```
int fast_stat_histogram (
            HISTOGRAM * histo,
            struct histstat * stbuf )
```

Fast and basic histogram statistics.

Compute mean and truncated mean for histogram. For this kind of histogram analysis actually no histogram is required. A 'moments' structure would be sufficient.

Parameters

histo	pointer to histogram (1-D)
stbuf	pointer to histogram statistics structure

Returns

Nonzero result indicates failure

References histogram::nbins_2d, histogram::tentries, and histogram::type.

7.28.3.16 fill_2d_int_histogram()

```
int fill_2d_int_histogram (
            HISTOGRAM * histo,
            long xvalue,
            long yvalue )
```

Increment a bin of a 2-D 'int' histogram by one.

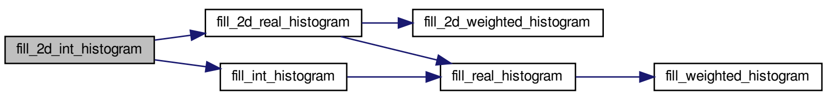
Increment a bin of a 2-D histogram by one. Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Arguments: histo – pointer to histogram xvalue, yvalue – X and Y positions where an entry is to be to the histogram (they may be outside the given ranges)

Return value: 0 (o.k.), -1 (no histogram that can be filled)

References fill_2d_real_histogram(), fill_int_histogram(), histogram::nbins_2d, and histogram::type.

Here is the call graph for this function:



7.28.3.17 fill_2d_real_histogram()

```
int fill_2d_real_histogram (
            HISTOGRAM * histo,
            double xvalue,
            double yvalue )
```

Increment a bin of a 2-D 'real' histogram by one.

Increment a bin of a 2-D histogram by one. Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Parameters

histo	Pointer to histogram
xvalue	X position where an entry is to be to the histogram (may be outside the given ranges)
yvalue	Y position where an entry is to be to the histogram (may be outside the given ranges)

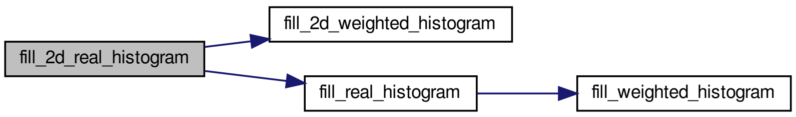
Returns

0 (o.k.), -1 (no histogram that can be filled)

References fill_2d_weighted_histogram(), fill_real_histogram(), histogram::nbins_2d, and histogram::type.

Referenced by fill_2d_int_histogram().

Here is the call graph for this function:



7.28.3.18 fill_2d_weighted_histogram()

```
int fill_2d_weighted_histogram (
             HISTOGRAM * histo,
             double xvalue,
             double yvalue,
             double weight )
```

Add an entry to a weighted 2-D histogram.

Increment a bin of a 2-D histogram by a given weight rather than by 1. This requires a suitable histogram type 'F' or 'D'.

Parameters

histo	Pointer to histogram.
xvalue	X posistion where an entry is to be added.
yvalue	Y posistion where an entry is to be added.
weight	The weight of that entry.

Returns

0 (o.k.), -1 (no histogram that can be filled with weights)

References histogram::ident, and histogram::type.

Referenced by fill_2d_real_histogram().

7.28.3.19 fill_histogram()

```
int fill_histogram (
            HISTOGRAM * histo,
            double xvalue,
            double yvalue,
            double weight )
```

Fill any type of 1-D or 2-D histogram known by its pointer.

Generic histogram fill function that can be used for type 'I', 'R', 'F', and 'D' histograms, although it is not recommended for type 'I' histograms, due to type conversions.

Parameters

histo	Pointer to histogram.
xvalue	X posistion where an entry is to be added.
yvalue	Y posistion (ignored for 1-D histograms)
weight	The weight of that entry (must be 1.0 for 'I' and 'R' type histograms).

Returns

0 (o.k.), -1 (no histogram that can be filled)

References histogram::ident, and histogram::type.

7.28.3.20 fill_histogram_by_ident()

```
int fill_histogram_by_ident (
            long id,
            double xvalue,
            double yvalue,
            double weight )
```

Fill any type of 1-D or 2-D histogram known by its ID number.

Generic histogram fill function that can be used for type 'I', 'R', 'F', and 'D' histograms, although it is not recommended for type 'I' histograms, due to type conversions.

Parameters

id	Identifier number of the histogram.
xvalue	X posistion where an entry is to be added.
yvalue	Y posistion (ignored for 1-D histograms)
weight	The weight of that entry (must be 1.0 for 'I' and 'R' type histograms).

Returns

0 (o.k.), -1 (no histogram that can be filled)

7.28.3.21 fill_int_histogram()

```
int fill_int_histogram (
             HISTOGRAM * histo,
             long value )
```

Increment a bin of a 1-D 'int' histogram by one.

Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Parameters

histo	Pointer to histogram
value	Position where an entry is to be added (may be outside the given range)

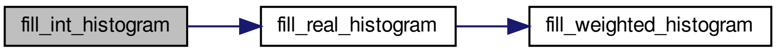
Returns

0 (o.k.), -1 (no histogram that can be filled)

References fill_real_histogram(), and histogram::type.

Referenced by fill_2d_int_histogram().

Here is the call graph for this function:



7.28.3.22 fill_real_histogram()

```
int fill_real_histogram (
             HISTOGRAM * histo,
             double value )
```

Increment a bin of a 1-D 'real' histogram by one.

Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Parameters

histo	Pointer to histogram
value	Position where an entry is to be added (may be outside the given range)

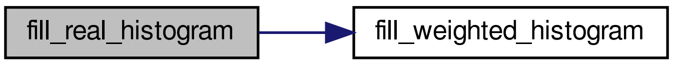
Returns

0 (o.k.), -1 (no histogram that can be filled)

References fill_weighted_histogram(), and histogram::type.

Referenced by fill_2d_real_histogram(), and fill_int_histogram().

Here is the call graph for this function:



7.28.3.23 fill_weighted_histogram()

```
int fill_weighted_histogram (
            HISTOGRAM * histo,
            double value,
            double weight )
```

Add an entry to a weighted 1-D histogram.

Increment a bin of a histogram by a given weight rather than by 1. This requires a suitable histogram type 'F' or 'D'.

Parameters

histo	Pointer to histogram.
value	Position where an entry is to be added.
weight	The weight of that entry.

Returns

0 (o.k.), -1 (no histogram that can be filled with weights)

References histogram::ident, and histogram::type.

Referenced by fill_real_histogram(), and project_histogram().

7.28.3.24 free_all_histograms()

```
void free_all_histograms (
             void  )
```

Deletes all histograms which are included in the linked list of histograms.

Returns

(none)

7.28.3.25 free_histo_contents()

```
static void free_histo_contents (
             HISTOGRAM * histo )  [static]
```

Free the contents (data pointers) of a histogram to be released or removed.

Parameters

Pointer	to histogram that should be 'cleaned'.

Returns

(none)

References histogram::counts, Histogram_Extension::ddata, histogram::extension, Histogram_Extension::fdata, and histogram::title.

7.28.3.26 free_histogram()

```
void free_histogram (
             HISTOGRAM * histo )
```

Free a histogram completely (both data and control structure).

Deallocates memory previously allocated to a histogram. If release_histogram was applied to that histogram before, it cannot be reallocated.

Parameters

histo	– pointer to previously allocated histogram

Returns

(none)

Referenced by project_histogram().

7.28.3.27 get_first_histogram()

```
HISTOGRAM* get_first_histogram (
             void   )
```

Get a pointer to the first histogram.

Get a pointer to the first histogram in the linked list of available histograms without making the corresponding variable global.

Returns

Pointer to the first histogram in the linked list.

Referenced by convert_histograms_to_root(), write_all_histograms(), and write_histograms().

7.28.3.28 get_histogram_by_ident()

```
HISTOGRAM* get_histogram_by_ident (
             long ident )
```

Get a histogram with the given ID.

Get the first histogram with a given ident (different from 0) or return NULL pointer if none exists.

Parameters

ident	– The histogram ident to be searched for.

Returns

Histogram pointer or NULL

Referenced by histogram_to_root(), project_histogram(), and write_dst_histos().

7.28.3.29 histogram_hashing()

```
int histogram_hashing (
             int tabsize )
```

Turn hashing of histograms (using their ident as key) on or off.

Parameters

tabsize	Minimum number of elements in hashing table or 0 if hash table should be released (max: 15000).

Returns

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

7.28.3.30 histogram_matching()

```
int histogram_matching (
            HISTOGRAM * histo1,
            HISTOGRAM * histo2 )
```

Check if two histograms have exactly matching definitions (same type, dimension, size, ranges).

Parameters

histo1	pointer to first histogram
histo2	pointer to second histogram

Returns

0 (not matching) or 1 (matching)

References histogram::counts, histogram::extension, Histogram_Parameters::integer, Histogram_Parameters↩ ::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, histogram::type, and Histogram↩ _Parameters::upper_limit.

7.28.3.31 histogram_to_lookup()

```
int histogram_to_lookup (
            HISTOGRAM * histo,
            HISTOGRAM * lookup )
```

Convert a histogram to a lookup table by integrating the histogram.

Parameters

histo	input histogram
lookup	output lookup table

Returns

0 if ok or -1 for failure

7.28.3.32 list_histograms()

```
void list_histograms (
            long ident )
```

List all available histograms using the 'Output()' function.

Parameters

ident	– histogram ident to search or 0

Returns

(none)

7.28.3.33 locate_histogram_fraction()

```
double locate_histogram_fraction (
            HISTOGRAM * histo,
            double fraction )
```

Locate point of arbitrary fraction of entries (quantile).

Locate the place in a 1-D histogram where a given fraction of the entries is to the 'left' of this place ('I' and 'R' type only).

Parameters

histo	Pointer to histogram
fraction	Fraction of entries to the left.

Returns

x-coordinate of given fraction or 0. for error.

7.28.3.34 lookup_int()

```
long lookup_int (
            HISTOGRAM * lookup,
            long value,
            long factor )
```

Look up a table created from an integer histogram.

Parameters

lookup	the lookup table
value	the value at which to look up
factor	the scaling factor of the lookup result or 0

Returns

If 'value' is inside the range of the lookup table (that is the range of the histogram from which the lookup table was created), a value between 0 and 'factor' (or the number of entries in the range, if factor==0) is returned.

References histogram::counts, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram↩ ::nbins, histogram::nbins_2d, histogram::tentries, histogram::type, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

7.28.3.35 lookup_real()

```
double lookup_real (
            HISTOGRAM * lookup,
            double value,
            double factor )
```

Look up a table created from an 'real' histogram.

Parameters

lookup	the lookup table
value	the value at which to look up
factor	the scaling factor of the lookup result or 0

Returns

If 'value' is inside the range of the lookup table (that is the range of the histogram from which the lookup table was created), a value between 0 and 'factor' (or the number of entries in the range, if factor==0) is returned.

References histogram::counts, Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, histogram::tentries, histogram::type, and Histogram_Parameters::upper_limit.

7.28.3.36 print_histogram()

```
void print_histogram (
            HISTOGRAM * histo )
```

Print contents of a histogram on the terminal.

Showing the actual content of each bin.

Parameters

histo	Pointer to histogram

Returns

(none)

References histogram::counts, histogram::extension, Histogram_Parameters::integer, Histogram_Parameters↩ ::lower_limit, histogram::nbins, Histogram_Parameters::real, histogram::tentries, histogram::type, and Histogram↩ _Parameters::upper_limit.

7.28.3.37 print_histogram_scaled()

```
void print_histogram_scaled (
            HISTOGRAM * histo,
            double fact )
```

Print scaled contents of a histogram on the terminal.

Showing the actual content of each bin. Only supported for types 'F' (float) and 'D' (double).

Parameters

histo	Pointer to histogram fact Scaling factor or zero for normalized.

Returns

(none)

References histogram::extension, histogram::ident, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, histogram::tentries, histogram::type, and Histogram_↩ Parameters::upper_limit.

7.28.3.38 set_first_histogram()

```
void set_first_histogram (
            HISTOGRAM * new_first_histogram )
```

Set a new histogram as the first element (context switching).

To allow 'context switching' of histograms the first element of the linked list of histograms can be changed by this function. Before that, the old value should be obtained with get_first_histogram() and saved. Note: For context switching it is not necessary to specify the actually first member of a linked list but any member of a list can be specifed to activate that list.

Parameters

new_first_histogram	A histogram in the new list (may be NULL pointer).

Returns

none

7.28.3.39 sort_histograms()

```
void sort_histograms (
             void )
```

Sort histograms in linked list by idents.

Returns

(none)

7.28.3.40 stat_histogram()

```
int stat_histogram (
             HISTOGRAM * histo,
             struct histstat * stbuf )
```

Statistical analysis of a histogram.

The median calculation is implemented for 1-D 'I' and 'R' types histograms only.

Parameters

histo	pointer to histogram
stbuf	pointer to histogram statistics structure

Returns

Nonzero result indicates failure

7.28.3.41 unlink_histogram()

```
void unlink_histogram (
            HISTOGRAM * histo )
```

Remove a histogram from the list without destroying it.

Remove a histogram from the linked list of histograms. That histogram will therefore not be found by any subsequent call to 'free_all_histograms()', display_all_histograms()', and 'get_histogram_by_ident()'.

Parameters

histo	Pointer to histogram.

Returns

(none)

Referenced by project_histogram().

7.28.4 Variable Documentation

7.28.4.1 primetab

```
CONST_QUAL short primetab[ ]  [static]
```

Initial value:

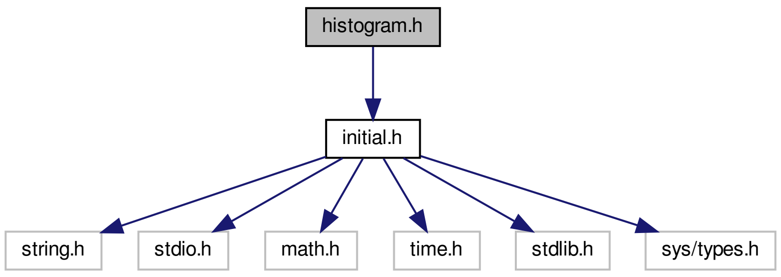
```
=
    { 131, 233, 353, 541, 751, 1051, 1367, 1511, 1723,
      1931, 2393, 3163, 3907, 5261, 6143, 7187, 8623, 9749, 11321, 15031 }
```

7.29 histogram.h File Reference

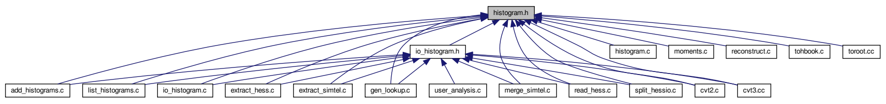
Declarations for handling one- and two-dimensional histograms.

`#include "initial.h"`

Include dependency graph for histogram.h:



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



Data Structures

- union Histogram_Parameters

  Parameters defining the usable range of coordinates.
- struct Histogram_Extension

  A histogram extension only allocated for weighted histograms.
- struct histogram

  A complete 1-D or 2-D histogram with control and data elements.
- struct histstat

  Statistics element for histogram analysis.
- struct momstat

  First, second, and higher moments of a 1-D histogram.
- struct moments

  Numbers to be summed up to obtain the moments.

Macros

- #define **MAX_HISTCOUNT** 4294967295UL /∗ or ULONG_MAX from <limits.h> ∗/

Typedefs

- typedef double HISTVALUE_REAL

  May be 'float' for ANSI C compiler.
- typedef long HISTVALUE_INT

  Short int is not recommended.
- typedef unsigned long HISTCOUNT

  The histogram counts may be unsigned short or unsigned long.
- typedef double HISTSUM_REAL

  To avoid loss of precision for adding many numbers, sums are of double type if 'real' type HISTVALUEs are used.
- typedef long **HISTSUM_INT**
- typedef double **HISTSTATVALUE**
- typedef struct histogram **HISTOGRAM**
- typedef struct moments **MOMENTS**

Functions

- void **histogram_lock** (HISTOGRAM ∗histo)
- void **histogram_unlock** (HISTOGRAM ∗histo)
- HISTOGRAM ∗ get_first_histogram (void)

    Get a pointer to the first histogram.
- void set_first_histogram (HISTOGRAM ∗new_first_histogram)

    Set a new histogram as the first element (context switching).
- HISTOGRAM ∗ get_histogram_by_ident (long ident)

    Get a histogram with the given ID.
- void list_histograms (long ident)

    List all available histograms using the 'Output()' function.
- HISTOGRAM ∗ book_histogram (long id, const char ∗title, const char ∗type, int dimension, double ∗low, double ∗high, int ∗nbins)

    General histogram booking function, assigning ID and title.
- HISTOGRAM ∗ book_int_histogram (long id, const char ∗title, int dimension, long ∗low, long ∗high, int ∗nbins)

    Book and integer-type histogram (content incremented by one per entry).
- HISTOGRAM ∗ book_1d_histogram (long id, const char ∗title, const char ∗type, double low, double high, int nbins)

    Simplified histogram booking function for one-dimensional histograms, assigning ID and title.
- HISTOGRAM ∗ allocate_histogram (const char ∗type, int dimension, double ∗low, double ∗high, int ∗nbins)

    Allocate any histogram without ID and title.
- HISTOGRAM ∗ alloc_int_histogram (long low, long high, int nbins)

    Allocate memory for a 1-D 'int' histogram and initialize it.
- HISTOGRAM ∗ alloc_real_histogram (double low, double high, int nbins)

    Allocate memory for a 1-D 'real' histogram and initialize it.
- HISTOGRAM ∗ alloc_2d_int_histogram (long xlow, long xhigh, int nxbins, long ylow, long yhigh, int nybins)

    Allocate memory for a 2-D 'int' histogram and initialize it.
- HISTOGRAM ∗ alloc_2d_real_histogram (double xlow, double xhigh, int nxbins, double ylow, double yhigh, int nybins)

    Allocate memory for a 2-D 'int' histogram and initialize it.
- void describe_histogram (HISTOGRAM ∗histo, const char ∗title, long ident)

    Add a describing title to a histogram previously allocated.
- void clear_histogram (HISTOGRAM ∗histo)

    Initialize an existing histogram.
- void free_histogram (HISTOGRAM ∗histo)

    Free a histogram completely (both data and control structure).
- void free_all_histograms (void)

    Deletes all histograms which are included in the linked list of histograms.
- void unlink_histogram (HISTOGRAM ∗histo)

    Remove a histogram from the list without destroying it.
- int fill_int_histogram (HISTOGRAM ∗histo, long value)

    Increment a bin of a 1-D 'int' histogram by one.
- int fill_real_histogram (HISTOGRAM ∗histo, double value)

    Increment a bin of a 1-D 'real' histogram by one.
- int fill_weighted_histogram (HISTOGRAM ∗histo, double value, double weight)

    Add an entry to a weighted 1-D histogram.
- int fill_2d_int_histogram (HISTOGRAM ∗histo, long xvalue, long yvalue)

    Increment a bin of a 2-D 'int' histogram by one.
- int fill_2d_real_histogram (HISTOGRAM ∗histo, double xvalue, double yvalue)

    Increment a bin of a 2-D 'real' histogram by one.

- int fill_2d_weighted_histogram (HISTOGRAM ∗histo, double xvalue, double yvalue, double weight)

  Add an entry to a weighted 2-D histogram.
- int fill_histogram (HISTOGRAM ∗histo, double xvalue, double yvalue, double weight)

  Fill any type of 1-D or 2-D histogram known by its pointer.
- int fill_histogram_by_ident (long id, double xvalue, double yvalue, double weight)

  Fill any type of 1-D or 2-D histogram known by its ID number.
- int stat_histogram (HISTOGRAM ∗histo, struct histstat ∗stbuf)

  Statistical analysis of a histogram.
- double locate_histogram_fraction (HISTOGRAM ∗histo, double fraction)

  Locate point of arbitrary fraction of entries (quantile).
- int fast_stat_histogram (HISTOGRAM ∗histo, struct histstat ∗stbuf)

  Fast and basic histogram statistics.
- int histogram_matching (HISTOGRAM ∗histo1, HISTOGRAM ∗histo2)

  Check if two histograms have exactly matching definitions (same type, dimension, size, ranges).
- HISTOGRAM ∗ add_histogram (HISTOGRAM ∗histo1, HISTOGRAM ∗histo2)

  Add a second histogram to a first one.
- void print_histogram_scaled (HISTOGRAM ∗histo, double fact)

  Print scaled contents of a histogram on the terminal.
- void print_histogram (HISTOGRAM ∗histo)

  Print contents of a histogram on the terminal.
- void display_histogram (HISTOGRAM ∗histo)

  Display contents of a histogram on the terminal.
- void display_all_histograms (void)

  Display all histograms in list of histograms.
- int histogram_to_lookup (HISTOGRAM ∗histo, HISTOGRAM ∗lookup)

  Convert a histogram to a lookup table by integrating the histogram.
- long lookup_int (HISTOGRAM ∗lookup, long value, long factor)

  Look up a table created from an integer histogram.
- double lookup_real (HISTOGRAM ∗lookup, double value, double factor)

  Look up a table created from an 'real' histogram.
- int histogram_hashing (int tabsize)

  Turn hashing of histograms (using their ident as key) on or off.
- void sort_histograms (void)

  Sort histograms in linked list by idents.
- void **release_histogram** (HISTOGRAM ∗histo)
- MOMENTS ∗ alloc_moments (double low, double high)

  Allocate a structure for sums of powers of data.
- void clear_moments (MOMENTS ∗mom)

  Initialize an existing moments structure (except for its range limits).
- void free_moments (MOMENTS ∗mom)

  Deallocates memory previously allocated to a moments structure.
- void fill_moments (MOMENTS ∗mom, double value)

  Add up those things needed to compute mean, standard deviation, skewness, and kurtosis (both for all data and separately for data in a range defined in alloc_moments().
- void fill_mean (MOMENTS ∗mom, double value)

  Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().
- void fill_mean_and_sigma (MOMENTS ∗mom, double value)

  Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().
- void fill_real_moments (MOMENTS ∗mom, double value, double weight)

Add up those things needed to compute – mean, – standard deviation, – skewness, and – kurtosis (both for all data and separately for data in a range defined in alloc_moments().

- void fill_real_mean (MOMENTS ∗mom, double value, double weight)

  Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().
- void fill_real_mean_and_sigma (MOMENTS ∗mom, double value, double weight)

  Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().
- int stat_moments (MOMENTS ∗mom, struct momstat ∗stmom)

  Calculate moments (mean, rms, skewness, kurtosis) from the sums of powers of data values.

7.29.1 Detailed Description

Declarations for handling one- and two-dimensional histograms.

The functions to work with these histograms is found in histogram.c . Eventio routines are available in io_histogram.c and conversion to HBOOK format is available through the 'cvt2' program. Handling of moments of a 1-D distribution is implemented in moments.c .

Author

Konrad Bernloehr

Date

1991 - 2023

7.29.2 Typedef Documentation

7.29.2.1 HISTCOUNT

```
typedef unsigned long HISTCOUNT
```

The histogram counts may be unsigned short or unsigned long.

With a unsigned short the overflow of a bin might easily happen.

7.29.2.2 HISTVALUE_REAL

```
typedef double HISTVALUE_REAL
```

May be 'float' for ANSI C compiler.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
For compatibility reasons the following 'typedef's are kept, but
the defined types should not be used any more because all of them
were changed in histogram.c to 'long', 'double', etc.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

HISTVALUE may be either an 'integer' type (recommended: long int) or a 'real' type (recommended: double). The method of calculating the array index corresponding to a given value is somewhat different for these two alternatives. Using a float for the 'real' type instead of a double would make no difference. However, a short int or an unsigned short int as 'integer' type requires more care for the calculation of the array index compared to a long or a unsigned long (frequent overflows unless a type cast of intermediate values to a long type is used).

7.29.3 Function Documentation

7.29.3.1 add_histogram()

```
HISTOGRAM* add_histogram (
             HISTOGRAM * histo1,
             HISTOGRAM * histo2 )
```

Add a second histogram to a first one.

The histograms must exactly match in their definitions. The first histogram will be modified, the second is unchanged.

Parameters

histo1	pointer to first histogram
histo2	pointer to second histogram

Returns

NULL pointer indicates failure.

7.29.3.2 alloc_2d_int_histogram()

```
HISTOGRAM* alloc_2d_int_histogram (
             long xlow,
             long xhigh,
             int nxbins,
             long ylow,
             long yhigh,
             int nybins )
```

Allocate memory for a 2-D 'int' histogram and initialize it.

Resulting histogram has integer range limits and integer contents (incremented by one per entry).

Parameters

xlow	lower limit of values in X to be covered by histogram
xhigh	upper limit ...
nxbins	the number of bins to be allocated in X
ylow	lower limit of values in Y to be covered by histogram
yhigh	upper limit ...
nybins	the number of bins to be allocated in Y

Returns

pointer to allocated histogram or NULL

References aux_alloc_histogram().

Here is the call graph for this function:

alloc_2d_int_histogram
aux_alloc_histogram

7.29.3.3 alloc_2d_real_histogram()

```
HISTOGRAM* alloc_2d_real_histogram (
            double xlow,
            double xhigh,
            int nxbins,
            double ylow,
            double yhigh,
            int nybins )
```

Allocate memory for a 2-D 'int' histogram and initialize it.

Resulting histogram has floating point range limits and integer contents (incremented by one per entry).

Parameters

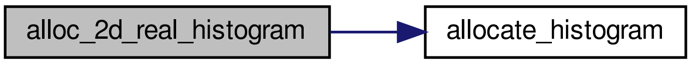
xlow	lower limit of values in X to be covered by histogram
xhigh	upper limit ...
nxbins	the number of bins to be allocated in X
ylow	lower limit of values in Y to be covered by histogram
yhigh	upper limit ...
nybins	the number of bins to be allocated in Y

Returns

pointer to allocated histogram or NULL

References allocate_histogram().

Here is the call graph for this function:



7.29.3.4 alloc_int_histogram()

```
HISTOGRAM* alloc_int_histogram (
             long low,
             long high,
             int nbins )
```

Allocate memory for a 1-D 'int' histogram and initialize it.

Resulting histogram has integer range limits and integer contents (incremented by one per entry).

Parameters

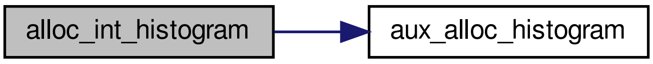
low	lower limit of values to be covered by histogram
high	upper limit ...
nbins	the number of bins to be allocated

Returns

pointer to allocated histogram or NULL

References aux_alloc_histogram().

Here is the call graph for this function:



7.29.3.5 alloc_moments()

```
MOMENTS* alloc_moments (
             HISTVALUE_REAL low,
             HISTVALUE_REAL high )
```

Allocate a structure for sums of powers of data.

Returns NULL if no structure could be allocated.

Parameters

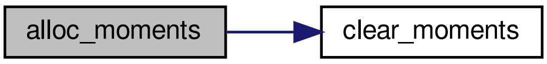
low	Lower limit of range for truncation
high	Upper limit of range for truncation

Returns

Pointer to allocated structure or NULL.

References clear_moments().

Here is the call graph for this function:



7.29.3.6 alloc_real_histogram()

```
HISTOGRAM* alloc_real_histogram (
             double low,
             double high,
             int nbins )
```

Allocate memory for a 1-D 'real' histogram and initialize it.

Resulting histogram has floating point range limits and integer contents (incremented by one per entry).

Parameters

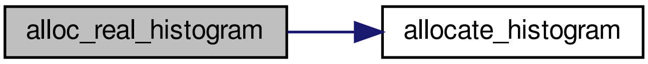
low	lower limit of values to be covered by histogram
high	upper limit ...
nbins	the number of bins to be allocated

Returns

pointer to allocated histogram or NULL

References allocate_histogram().

Here is the call graph for this function:



7.29.3.7 allocate_histogram()

```
HISTOGRAM* allocate_histogram (
             const char * type,
             int dimension,
             double * low,
             double * high,
             int * nbins )
```

Allocate any histogram without ID and title.

Allocate a histogram of 1 or 2 dimensions, 'I', 'R', 'F' or 'D' type, without assigning an ID number and title string to it. To avoid the (long) <--> (double) typecasts, the direct calls to alloc_int_histogram() and alloc_2d_int_histogram() are recommended for integer-limits histograms (type 'I').

Parameters

type	"I" (int, no weights), "R" (real, no weights), "F" (float, with weights), "D" (double, w.w.)
dimension	1 or 2 for 1-D or 2-D histogram
low	Pointer to lower limits (x or x,y for 1-D or 2-D)
high	Pointer to upper limits
nbins	Pointer to no. of bins per dimension (nx or nx, ny)

Returns

Pointer to new histogram or NULL

Referenced by alloc_2d_real_histogram(), alloc_real_histogram(), book_1d_histogram(), and book_histogram().

7.29.3.8 book_1d_histogram()

```
HISTOGRAM* book_1d_histogram (
            long id,
            const char * title,
            const char * type,
            double low,
            double high,
            int nbins )
```

Simplified histogram booking function for one-dimensional histograms, assigning ID and title.

Book a histogram of one dimension, 'I', 'R', 'F', or 'D' type. The histogram is allocated (if possible) and the supplied ID number and title string are assigned.

Parameters

id	ID number
title	Histogram title string
type	"I" (int, no weights), "R" (real, no weights), "F" (float, with weights), "D" (double, w.w.)
low	Lower limit (x)
high	Upper limit (x)
nbins	No. of bins (nx)

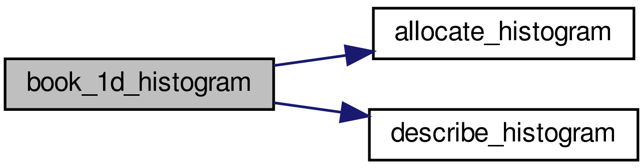
Returns

Pointer to new histogram or NULL

References allocate_histogram(), and describe_histogram().

Referenced by project_histogram().

Here is the call graph for this function:



7.29.3.9 book_histogram()

```
HISTOGRAM* book_histogram (
             long id,
             const char * title,
             const char * type,
             int dimension,
             double * low,
             double * high,
             int * nbins )
```

General histogram booking function, assigning ID and title.

Book a histogram of 1 or 2 dimensions, 'I', 'R', 'F', or 'D' type. The histogram is allocated (if possible) and the supplied ID number and title string are assigned.

Parameters

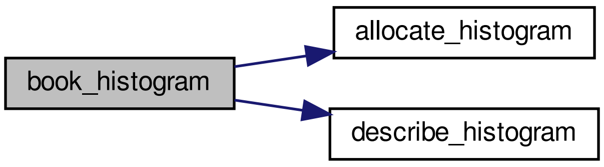
id	ID number
title	Histogram title string
type	"I" (int, no weights), "R" (real, no weights), "F" (float, with weights), "D" (double, w.w.)
dimension	1 or 2 for 1-D or 2-D histogram
low	Pointer to lower limits (x or x,y for 1-D or 2-D)
high	Pointer to upper limits
nbins	Pointer to no. of bins per dimension (nx or nx, ny)

Returns

Pointer to new histogram or NULL

References allocate_histogram(), and describe_histogram().

Here is the call graph for this function:



7.29.3.10 book_int_histogram()

```
HISTOGRAM* book_int_histogram (
             long id,
             const char * title,
             int dimension,
             long * low,
             long * high,
             int * nbins )
```

Book and integer-type histogram (content incremented by one per entry).

Like book_histogram() but for 'I' type histograms only (1-D or 2-D)

Parameters

id	ID number
title	Histogram title string
dimension	1 or 2 for 1-D or 2-D histogram
low	Pointer to lower limits (x or x,y for 1-D or 2-D)
high	Pointer to upper limits
nbins	Pointer to no. of bins per dimension (nx or nx, ny)

Returns

Pointer to new histogram or NULL

7.29.3.11 clear_histogram()

```
void clear_histogram (
             HISTOGRAM * histo )
```

Initialize an existing histogram.

Parameters

histo	– pointer to histogram

Returns

(none)

Referenced by write_dst_histos().

7.29.3.12 clear_moments()

```
void clear_moments (
            MOMENTS * mom )
```

Initialize an existing moments structure (except for its range limits).

Parameters

mom	Pointer to moments structure

Referenced by alloc_moments().

7.29.3.13 describe_histogram()

```
void describe_histogram (
            HISTOGRAM * histo,
            const char * title,
            long ident )
```

Add a describing title to a histogram previously allocated.

Parameters

histo	Histogram to which the title should be added
title	The title string. This is ignored if the histogram already has a title.
ident	Identification number, must be unique (or 0) if any I/O is intended, because read_histogram() deletes a pre-existing histogram with the same ID.

Returns

none

Referenced by book_1d_histogram(), and book_histogram().

7.29.3.14 display_all_histograms()

```
void display_all_histograms (
            void )
```

Display all histograms in list of histograms.

Arguments: none

Return value: none

7.29.3.15 display_histogram()

```
void display_histogram (
             HISTOGRAM * histo )
```

Display contents of a histogram on the terminal.

This is a simple 'HPRINT' type display on one screen.

Parameters

histo	Pointer to histogram

Returns

(none)

References histogram::counts, histogram::extension, histogram::nbins, histogram::tentries, and histogram::type.

7.29.3.16 fast_stat_histogram()

```
int fast_stat_histogram (
             HISTOGRAM * histo,
             struct histstat * stbuf )
```

Fast and basic histogram statistics.

Compute mean and truncated mean for histogram. For this kind of histogram analysis actually no histogram is required. A 'moments' structure would be sufficient.

Parameters

histo	pointer to histogram (1-D)
stbuf	pointer to histogram statistics structure

Returns

Nonzero result indicates failure

References histogram::nbins_2d, histogram::tentries, and histogram::type.

7.29.3.17 fill_2d_int_histogram()

```
int fill_2d_int_histogram (
             HISTOGRAM * histo,
             long xvalue,
             long yvalue )
```

Increment a bin of a 2-D 'int' histogram by one.

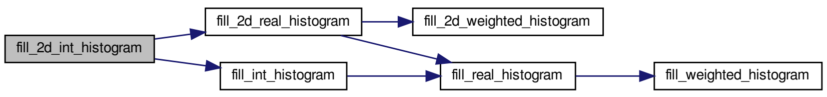
Increment a bin of a 2-D histogram by one. Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Arguments: histo – pointer to histogram xvalue, yvalue – X and Y positions where an entry is to be to the histogram (they may be outside the given ranges)

Return value: 0 (o.k.), -1 (no histogram that can be filled)

References fill_2d_real_histogram(), fill_int_histogram(), histogram::nbins_2d, and histogram::type.

Here is the call graph for this function:



7.29.3.18 fill_2d_real_histogram()

```
int fill_2d_real_histogram (
            HISTOGRAM * histo,
            double xvalue,
            double yvalue )
```

Increment a bin of a 2-D 'real' histogram by one.

Increment a bin of a 2-D histogram by one. Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Parameters

histo	Pointer to histogram
xvalue	X position where an entry is to be to the histogram (may be outside the given ranges)
yvalue	Y position where an entry is to be to the histogram (may be outside the given ranges)

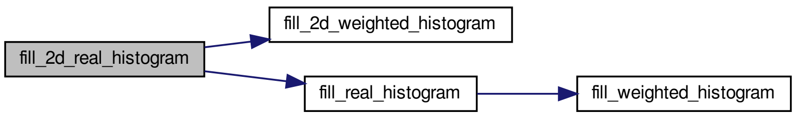
Returns

0 (o.k.), -1 (no histogram that can be filled)

References fill_2d_weighted_histogram(), fill_real_histogram(), histogram::nbins_2d, and histogram::type.

Referenced by fill_2d_int_histogram().

Here is the call graph for this function:



7.29.3.19 fill_2d_weighted_histogram()

```
int fill_2d_weighted_histogram (
            HISTOGRAM * histo,
            double xvalue,
            double yvalue,
            double weight )
```

Add an entry to a weighted 2-D histogram.

Increment a bin of a 2-D histogram by a given weight rather than by 1. This requires a suitable histogram type 'F' or 'D'.

Parameters

histo	Pointer to histogram.
xvalue	X posistion where an entry is to be added.
yvalue	Y posistion where an entry is to be added.
weight	The weight of that entry.

Returns

0 (o.k.), -1 (no histogram that can be filled with weights)

References histogram::ident, and histogram::type.

Referenced by fill_2d_real_histogram().

7.29.3.20 fill_histogram()

```
int fill_histogram (
            HISTOGRAM * histo,
            double xvalue,
            double yvalue,
            double weight )
```

Fill any type of 1-D or 2-D histogram known by its pointer.

Generic histogram fill function that can be used for type 'I', 'R', 'F', and 'D' histograms, although it is not recommended for type 'I' histograms, due to type conversions.

Parameters

histo	Pointer to histogram.
xvalue	X posistion where an entry is to be added.
yvalue	Y posistion (ignored for 1-D histograms)
weight	The weight of that entry (must be 1.0 for 'I' and 'R' type histograms).

Returns

0 (o.k.), -1 (no histogram that can be filled)

References histogram::ident, and histogram::type.

7.29.3.21 fill_histogram_by_ident()

```
int fill_histogram_by_ident (
            long id,
            double xvalue,
            double yvalue,
            double weight )
```

Fill any type of 1-D or 2-D histogram known by its ID number.

Generic histogram fill function that can be used for type 'I', 'R', 'F', and 'D' histograms, although it is not recommended for type 'I' histograms, due to type conversions.

Parameters

id	Identifier number of the histogram.
xvalue	X posistion where an entry is to be added.
yvalue	Y posistion (ignored for 1-D histograms)
weight	The weight of that entry (must be 1.0 for 'I' and 'R' type histograms).

Returns

0 (o.k.), -1 (no histogram that can be filled)

7.29.3.22 fill_int_histogram()

```
int fill_int_histogram (
            HISTOGRAM * histo,
            long value )
```

Increment a bin of a 1-D 'int' histogram by one.

Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Parameters

histo	Pointer to histogram
value	Position where an entry is to be added (may be outside the given range)

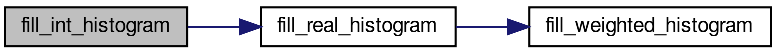
Returns

0 (o.k.), -1 (no histogram that can be filled)

References fill_real_histogram(), and histogram::type.

Referenced by fill_2d_int_histogram().

Here is the call graph for this function:



7.29.3.23 fill_mean()

```
void fill_mean (
            MOMENTS * mom,
            HISTVALUE_REAL value )
```

Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value

7.29.3.24 fill_mean_and_sigma()

```
void fill_mean_and_sigma (
            MOMENTS * mom,
            HISTVALUE_REAL value )
```

Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value

7.29.3.25 fill_moments()

```
void fill_moments (
            MOMENTS * mom,
            HISTVALUE_REAL value )
```

Add up those things needed to compute mean, standard deviation, skewness, and kurtosis (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value

7.29.3.26 fill_real_histogram()

```
int fill_real_histogram (
            HISTOGRAM * histo,
            double value )
```

Increment a bin of a 1-D 'real' histogram by one.

Either a count for one of the bins in the histogram range is incremented or an underflow or overflow count. For the calculation of the mean value and truncated mean value sums of values and number of histogram entries are updated as well.

Parameters

histo	Pointer to histogram
value	Position where an entry is to be added (may be outside the given range)

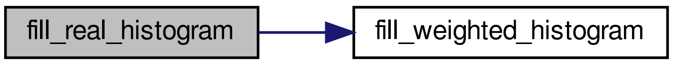
Returns

0 (o.k.), -1 (no histogram that can be filled)

References fill_weighted_histogram(), and histogram::type.

Referenced by fill_2d_real_histogram(), and fill_int_histogram().

Here is the call graph for this function:



7.29.3.27 fill_real_mean()

```
void fill_real_mean (
            MOMENTS * mom,
            HISTVALUE_REAL value,
            double weight )
```

Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value
weight	Weighting factor of this value

7.29.3.28 fill_real_mean_and_sigma()

```
void fill_real_mean_and_sigma (
            MOMENTS * mom,
            HISTVALUE_REAL value,
            double weight )
```

Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value
weight	Weighting factor of this value

7.29.3.29 fill_real_moments()

```
void fill_real_moments (
            MOMENTS * mom,
            HISTVALUE_REAL value,
            double weight )
```

Add up those things needed to compute – mean, – standard deviation, – skewness, and – kurtosis (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value
weight	Weighting factor of this value

7.29.3.30 fill_weighted_histogram()

```
int fill_weighted_histogram (
            HISTOGRAM * histo,
            double value,
            double weight )
```

Add an entry to a weighted 1-D histogram.

Increment a bin of a histogram by a given weight rather than by 1. This requires a suitable histogram type 'F' or 'D'.

Parameters

histo	Pointer to histogram.
value	Position where an entry is to be added.
weight	The weight of that entry.

Returns

0 (o.k.), -1 (no histogram that can be filled with weights)

References histogram::ident, and histogram::type.

Referenced by fill_real_histogram(), and project_histogram().

7.29.3.31 free_all_histograms()

```
void free_all_histograms (
            void  )
```

Deletes all histograms which are included in the linked list of histograms.

Returns

(none)

7.29.3.32 free_histogram()

```
void free_histogram (
            HISTOGRAM * histo )
```

Free a histogram completely (both data and control structure).

Deallocates memory previously allocated to a histogram. If release_histogram was applied to that histogram before, it cannot be reallocated.

Parameters

histo	– pointer to previously allocated histogram

Returns

(none)

Referenced by project_histogram().

7.29.3.33 free_moments()

```
void free_moments (
            MOMENTS * mom )
```

Deallocates memory previously allocated to a moments structure.

Parameters

mom	Pointer to previously allocated structure

7.29.3.34 get_first_histogram()

```
HISTOGRAM* get_first_histogram (
             void  )
```

Get a pointer to the first histogram.

Get a pointer to the first histogram in the linked list of available histograms without making the corresponding variable global.

Returns

Pointer to the first histogram in the linked list.

Referenced by convert_histograms_to_root(), write_all_histograms(), and write_histograms().

7.29.3.35 get_histogram_by_ident()

```
HISTOGRAM* get_histogram_by_ident (
             long ident )
```

Get a histogram with the given ID.

Get the first histogram with a given ident (different from 0) or return NULL pointer if none exists.

Parameters

ident	– The histogram ident to be searched for.

Returns

Histogram pointer or NULL

Referenced by histogram_to_root(), project_histogram(), and write_dst_histos().

7.29.3.36 histogram_hashing()

```
int histogram_hashing (
             int tabsize )
```

Turn hashing of histograms (using their ident as key) on or off.

Parameters

tabsize	Minimum number of elements in hashing table or 0 if hash table should be released (max: 15000).

Returns

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

7.29.3.37 histogram_matching()

```
int histogram_matching (
            HISTOGRAM * histo1,
            HISTOGRAM * histo2 )
```

Check if two histograms have exactly matching definitions (same type, dimension, size, ranges).

Parameters

histo1	pointer to first histogram
histo2	pointer to second histogram

Returns

0 (not matching) or 1 (matching)

References histogram::counts, histogram::extension, Histogram_Parameters::integer, Histogram_Parameters↩ ::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, histogram::type, and Histogram↩ _Parameters::upper_limit.

7.29.3.38 histogram_to_lookup()

```
int histogram_to_lookup (
            HISTOGRAM * histo,
            HISTOGRAM * lookup )
```

Convert a histogram to a lookup table by integrating the histogram.

Parameters

histo	input histogram
lookup	output lookup table

Returns

0 if ok or -1 for failure

7.29.3.39 list_histograms()

```
void list_histograms (
            long ident )
```

List all available histograms using the 'Output()' function.

Parameters

ident	– histogram ident to search or 0

Returns

(none)

7.29.3.40 locate_histogram_fraction()

```
double locate_histogram_fraction (
            HISTOGRAM * histo,
            double fraction )
```

Locate point of arbitrary fraction of entries (quantile).

Locate the place in a 1-D histogram where a given fraction of the entries is to the 'left' of this place ('I' and 'R' type only).

Parameters

histo	Pointer to histogram
fraction	Fraction of entries to the left.

Returns

x-coordinate of given fraction or 0. for error.

7.29.3.41 lookup_int()

```
long lookup_int (
            HISTOGRAM * lookup,
            long value,
            long factor )
```

Look up a table created from an integer histogram.

Parameters

lookup	the lookup table
value	the value at which to look up
factor	the scaling factor of the lookup result or 0

Returns

If 'value' is inside the range of the lookup table (that is the range of the histogram from which the lookup table was created), a value between 0 and 'factor' (or the number of entries in the range, if factor==0) is returned.

References histogram::counts, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram↩ ::nbins, histogram::nbins_2d, histogram::tentries, histogram::type, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

7.29.3.42 lookup_real()

```
double lookup_real (
            HISTOGRAM * lookup,
            double value,
            double factor )
```

Look up a table created from an 'real' histogram.

Parameters

lookup	the lookup table
value	the value at which to look up
factor	the scaling factor of the lookup result or 0

Returns

If 'value' is inside the range of the lookup table (that is the range of the histogram from which the lookup table was created), a value between 0 and 'factor' (or the number of entries in the range, if factor==0) is returned.

References histogram::counts, Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, histogram::tentries, histogram::type, and Histogram_Parameters::upper_limit.

7.29.3.43 print_histogram()

```
void print_histogram (
            HISTOGRAM * histo )
```

Print contents of a histogram on the terminal.

Showing the actual content of each bin.

Parameters

histo	Pointer to histogram

Returns

(none)

References histogram::counts, histogram::extension, Histogram_Parameters::integer, Histogram_Parameters↩ ::lower_limit, histogram::nbins, Histogram_Parameters::real, histogram::tentries, histogram::type, and Histogram↩ _Parameters::upper_limit.

7.29.3.44 print_histogram_scaled()

```
void print_histogram_scaled (
            HISTOGRAM * histo,
            double fact )
```

Print scaled contents of a histogram on the terminal.

Showing the actual content of each bin. Only supported for types 'F' (float) and 'D' (double).

Parameters

histo	Pointer to histogram fact Scaling factor or zero for normalized.

Returns

(none)

References histogram::extension, histogram::ident, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, histogram::tentries, histogram::type, and Histogram_↩ Parameters::upper_limit.

7.29.3.45 set_first_histogram()

```
void set_first_histogram (
            HISTOGRAM * new_first_histogram )
```

Set a new histogram as the first element (context switching).

To allow 'context switching' of histograms the first element of the linked list of histograms can be changed by this function. Before that, the old value should be obtained with get_first_histogram() and saved. Note: For context switching it is not necessary to specify the actually first member of a linked list but any member of a list can be specifed to activate that list.

Parameters

new_first_histogram	A histogram in the new list (may be NULL pointer).

Returns

none

7.29.3.46 sort_histograms()

```
void sort_histograms (
            void )
```

Sort histograms in linked list by idents.

Returns

(none)

7.29.3.47 stat_histogram()

```
int stat_histogram (
            HISTOGRAM * histo,
            struct histstat * stbuf )
```

Statistical analysis of a histogram.

The median calculation is implemented for 1-D 'I' and 'R' types histograms only.

Parameters

histo	pointer to histogram
stbuf	pointer to histogram statistics structure

Returns

Nonzero result indicates failure

7.29.3.48 stat_moments()

```
int stat_moments (
            MOMENTS * mom,
            struct momstat * stmom )
```

Calculate moments (mean, rms, skewness, kurtosis) from the sums of powers of data values.

Parameters

mom	'moments' structure with the sums of the powers of data values (only 1st power if only mean to be calculated, also 2nd power if r.m.s. to be calculated, and also 3rd and 4th if skewness and kurtosis wanted.
stmom	Pointer to structure for computed moments

Returns

0 (o.k.), -1 and -2 (invalid data)

7.29.3.49 unlink_histogram()

```
void unlink_histogram (
            HISTOGRAM * histo )
```

Remove a histogram from the list without destroying it.

Remove a histogram from the linked list of histograms. That histogram will therefore not be found by any subsequent call to 'free_all_histograms()', display_all_histograms()', and 'get_histogram_by_ident()'.

Parameters

histo	Pointer to histogram.

Returns

(none)

Referenced by project_histogram().

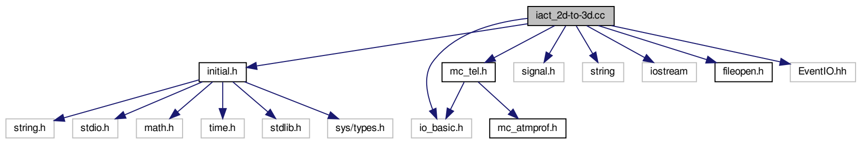
7.30 iact_2d-to-3d.cc File Reference

A program reading simulated CORSIKA data written through the IACT interface and converting photon bunches from the traditional format into the 3D format.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include <signal.h>
#include <string>
#include <iostream>
#include "fileopen.h"
```

```
#include "EventIO.hh"
```

Include dependency graph for iact_2d-to-3d.cc:



Macros

- #define **MAXTEL** 5

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- void **ioerrorcheck** (void)
- int tel_conv_mc_phot (EventIO &evio)

  Convert photon bunches from a single telescope.
- int array_conv_mc_phot (EventIO &evio)

  Convert photon bunches from a full array of telescopes.
- void **syntax** (const string &prg)
- int main (int argc, char $**$argv)

  Main program.

Variables

- static int **interrupted** = 0
- static int **verbose** = 0
- struct bunch $*$ **tel_bunches** [MAXTEL]
- struct bunch3d $*$ **tel_bunches3d** [MAXTEL]
- int **max_bunches** [MAXTEL]
- int **max_bunches3d** [MAXTEL]
- int **tel_nbunches** [MAXTEL]
- int **tel_nbunches3d** [MAXTEL]
- double **tel_photons** [MAXTEL]
- double **tel_photons3d** [MAXTEL]
- double **obslev** = 1835.e2
- double **zdet** [MAXTEL]

7.30.1 Detailed Description

A program reading simulated CORSIKA data written through the IACT interface and converting photon bunches from the traditional format into the 3D format.

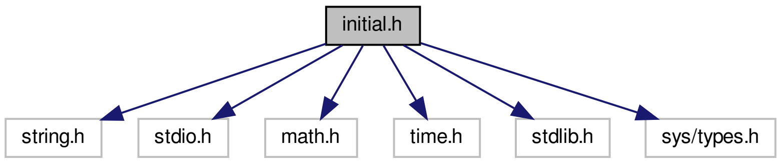
Since the conversion cannot recover the lack of support for horizontal and upward photons in the traditional format and since the 3D format needs more memory, the sole purpose of this tool is to serve for a cross-check of sim_↩ telarray with identical photons in the two formats.

7.31 initial.h File Reference

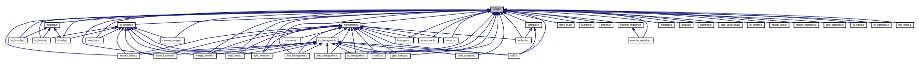
Indentification of the system and including some basic include file.

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

Typedefs

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

7.31.1 Detailed Description

Indentification of the system and including some basic include file.

Author

Konrad Bernloehr

Date

1991 to 2023

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.

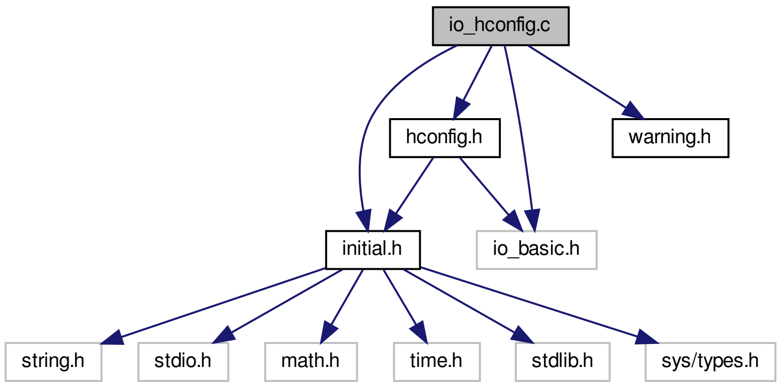
```
Identification of the host operating system (not CPU):

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
```

7.32 io_hconfig.c File Reference

Input and output of hconfig settings as EventIO data.

```
#include "initial.h"
#include "io_basic.h"
#include "hconfig.h"
#include "warning.h"
```

Include dependency graph for io_hconfig.c:



Macros

- #define **NO_INITIAL_MACROS** 1

Functions

- static int config_binary_write_as_text (IO_BUFFER ∗iobuf, char ∗text, int type)

  Put a hconfig name or text item into an I/O buffer.
- int config_binary_write_name (IO_BUFFER ∗iobuf, char ∗name)

  Write the name of a hconfig item for which binary data should follow.
- int config_binary_write_text (IO_BUFFER ∗iobuf, char ∗text)

  Write 'binary' hconfig data as text (for 'string' or 'function' types).
- int config_binary_read_text (IO_BUFFER ∗iobuf, char ∗name, int maxlen)

  Get a hconfig name or text item from an I/O buffer.
- int config_binary_text_length (IO_BUFFER ∗iobuf)

  If the next item is of the text type, get the length of the text.
- int config_binary_read_name (IO_BUFFER ∗iobuf, char ∗name, int maxlen)

  Is the same as config_binary_read_text().
- int config_binary_write_index (IO_BUFFER ∗iobuf, int nidx, int ∗idx_low, int ∗idx_high)

  Put a list of index ranges for binary hconfig data following.
- int config_binary_read_index (IO_BUFFER ∗iobuf, int ∗nidx, int ∗idx_low, int ∗idx_high, int max_idx)

  Get a list of index ranges for binary hconfig data following.
- int config_binary_envelope_begin (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

Begin with the envelope for a binary configuration item.
- int config_binary_envelope_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Close the envelope for a binary configuration item.
- int config_binary_inquire_numbers (IO_BUFFER ∗iobuf, int ∗ntype, int ∗nsize, int32_t ∗num, int ∗nopt)

  Tell me what kind of binary numbers follow in the next I/O item.
- int config_binary_read_numbers (IO_BUFFER ∗iobuf, void ∗data, size_t max_size)

  Get the binary numbers from the next I/O item.
- int config_binary_convert_data (void ∗out, int out_type, int out_size, void ∗in, int in_type, int in_size)

  Concert binary numbers of one type to numbers of another type.

7.32.1 Detailed Description

Input and output of hconfig settings as EventIO data.

Author

Konrad Bernloehr

Date

2001 to 2018

7.32.2 Function Documentation

7.32.2.1 config_binary_convert_data()

```
int config_binary_convert_data (
            void * out,
            int out_type,
            int out_size,
            void * in,
            int in_type,
            int in_size )
```

Concert binary numbers of one type to numbers of another type.

Supported types are signed integers of various lengths, unsigned integers of various lengths, float and double. The signed and unsigned integers can be 1, 2, 4 or perhaps 8 bytes long. Float should be 4 bytes long, double 8 bytes.

7.32.2.2 config_binary_read_text()

```
int config_binary_read_text (
            IO_BUFFER * iobuf,
            char * name,
            int maxlen )
```

Get a hconfig name or text item from an I/O buffer.

Both the IO_TYPE_HCONFIG_NAME and IO_TYPE_HCONFIG_TEXT eventio item types are simple text strings enclosed in an I/O item. Because either of them can appear at the beginning of binary configuration data (with different interpretations) they are distinguished by different item type numbers. Otherwise they are the same.

Referenced by config_binary_read_name().

7.32.2.3 config_binary_text_length()

```
int config_binary_text_length (
             IO_BUFFER * iobuf )
```

If the next item is of the text type, get the length of the text.

This allows finding out the length of the text first, allocating enough memory to read it and then start reading the text.

Returns

The length of the string not including the trailing '\0' which has to be appended.

7.32.2.4 config_binary_write_as_text()

```
static int config_binary_write_as_text (
             IO_BUFFER * iobuf,
             char * text,
             int type )  [static]
```

Put a hconfig name or text item into an I/O buffer.

Both the IO_TYPE_HCONFIG_NAME and IO_TYPE_HCONFIG_TEXT eventio item types are simple text strings enclosed in an I/O item. Because either of them can appear at the beginning of binary configuration data (with different interpretations) they are distinguished by different item type numbers. Otherwise they are the same.

Referenced by config_binary_write_name(), and config_binary_write_text().

7.32.2.5 config_binary_write_name()

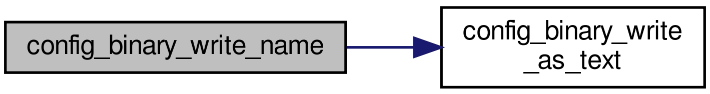
```
int config_binary_write_name (
             IO_BUFFER * iobuf,
             char * name )
```

Write the name of a hconfig item for which binary data should follow.

Calls config_binary_write_as_text().

References config_binary_write_as_text().

Here is the call graph for this function:



7.32.2.6 config_binary_write_text()

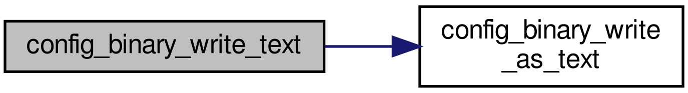
```
int config_binary_write_text (
            IO_BUFFER * iobuf,
            char * text )
```

Write 'binary' hconfig data as text (for 'string' or 'function' types).

Calls config_binary_write_as_text().

References config_binary_write_as_text().

Here is the call graph for this function:

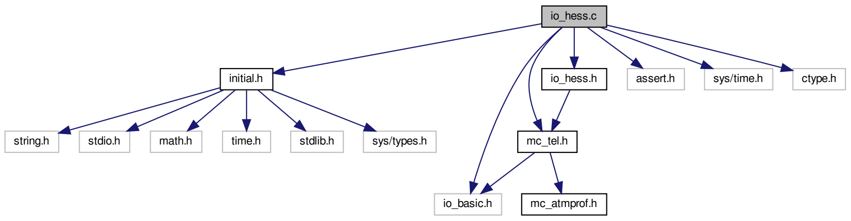


7.33 io_hess.c File Reference

Writing and reading of H.E.S.S.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_hess.h"
#include <assert.h>
#include <sys/time.h>
#include <ctype.h>
```

Include dependency graph for io_hess.c:



Functions

- void check_hessio_max (int ncheck, int max_tel, int max_pix, int max_sectors, int max_drawers, int max_↩ pixsectors, int max_slices, int max_hotpix, int max_profile, int max_d_temp, int max_c_temp, int max_gains)

  Support for checking if user functions are compiled with the same limits as the library.
- void **show_hessio_max** ()
- void hs_reset_env ()

  Allow user to override MAX_PRINT_ARRAY and PRINT_VERBOSE settings at a later time.
- static void hs_check_env ()

  Get settings on how much information to print from environment.
- static void put_time_blob (HTime ∗t, IO_BUFFER ∗iobuf)

  Put the time (seconds since 1970.0, nanoseconds) into an eventio block already started.
- static void get_time_blob (HTime ∗t, IO_BUFFER ∗iobuf)

  Get the time (seconds since 1970.0, nanoseconds) from an eventio block already started.
- void set_tel_idx_ref (int iref)

  Switch between multiple telescope lookup tables.
- void set_tel_idx (int ntel, int ∗idx)

  Setup of telescope index lookup table.
- int find_tel_idx (int tel_id)

  Lookup from telescope ID to offset number (index) in structures.
- int write_simtel_runheader (IO_BUFFER ∗iobuf, RunHeader ∗rh)

  Write the run header in eventio format.
- int read_simtel_runheader (IO_BUFFER ∗iobuf, RunHeader ∗rh)

  Read the run header in eventio format.
- int print_simtel_runheader (IO_BUFFER ∗iobuf)

  Read the run header in eventio format.
- int write_simtel_mcrunheader (IO_BUFFER ∗iobuf, MCRunHeader ∗mcrh)

  Write the Monte Carlo run header in eventio format.
- int read_simtel_mcrunheader (IO_BUFFER ∗iobuf, MCRunHeader ∗mcrh)

  Read the Monte Carlo run header in eventio format.
- int print_simtel_mcrunheader (IO_BUFFER ∗iobuf)

  Print the Monte Carlo run header data.
- int write_simtel_camsettings (IO_BUFFER ∗iobuf, CameraSettings ∗cs)

  Write the camera definition (pixel positions) in eventio format.
- int read_simtel_camsettings (IO_BUFFER ∗iobuf, CameraSettings ∗cs)

  Read the camera definition (pixel positions) in eventio format.
- int print_simtel_camsettings (IO_BUFFER ∗iobuf)

  Print the camera definition (pixel positions) in eventio format.
- int write_simtel_camorgan (IO_BUFFER ∗iobuf, CameraOrganisation ∗co)

  Write the logical organisation of camera electronics in eventio format.
- int read_simtel_camorgan (IO_BUFFER ∗iobuf, CameraOrganisation ∗co)

  Read the logical organisation of camera electronics in eventio format.
- int print_simtel_camorgan (IO_BUFFER ∗iobuf)

  Read the logical organisation of camera electronics in eventio format.
- int write_simtel_pixelset (IO_BUFFER ∗iobuf, PixelSetting ∗ps)

  Write the settings of pixel parameters (HV, thresholds, ...) in eventio format.
- int read_simtel_pixelset (IO_BUFFER ∗iobuf, PixelSetting ∗ps)

  Read the settings of pixel parameters (HV, thresholds, ...) in eventio format.
- int print_simtel_pixelset (IO_BUFFER ∗iobuf)

  Show the settings of pixel parameters (HV, thresholds, ...) in eventio format.
- int write_simtel_pixeldis (IO_BUFFER ∗iobuf, PixelDisabled ∗pd)

Write which pixels are disabled in HV and/or trigger in eventio format.
- int read_simtel_pixeldis (IO_BUFFER ∗iobuf, PixelDisabled ∗pd)

  Read which pixels are disabled in HV and/or trigger in eventio format.
- int print_simtel_pixeldis (IO_BUFFER ∗iobuf)

  Print which pixels are disabled in HV and/or trigger in eventio format.
- int write_simtel_camsoftset (IO_BUFFER ∗iobuf, CameraSoftSet ∗cs)

  Write camera software parameters relevant for data recording in eventio format.
- int read_simtel_camsoftset (IO_BUFFER ∗iobuf, CameraSoftSet ∗cs)

  Read camera software parameters relevant for data recording in eventio format.
- int write_simtel_trackset (IO_BUFFER ∗iobuf, TrackingSetup ∗ts)

  Write the settings for tracking of a telescope in eventio format.
- int read_simtel_trackset (IO_BUFFER ∗iobuf, TrackingSetup ∗ts)

  Read the settings for tracking of a telescope in eventio format.
- int print_simtel_trackset (IO_BUFFER ∗iobuf)

  Print the settings for tracking of a telescope in eventio format.
- int write_simtel_pointingcor (IO_BUFFER ∗iobuf, PointingCorrection ∗pc)

  Write the parameters of a telescope's pointing correction in eventio format.
- int read_simtel_pointingcor (IO_BUFFER ∗iobuf, PointingCorrection ∗pc)

  Read the parameters of a telescope's pointing correction in eventio format.
- int print_simtel_pointingcor (IO_BUFFER ∗iobuf)

  Print the parameters of a telescope's pointing correction in eventio format.
- int write_simtel_centralevent (IO_BUFFER ∗iobuf, CentralEvent ∗ce)

  Write the trigger data of the central trigger in eventio format.
- int read_simtel_centralevent (IO_BUFFER ∗iobuf, CentralEvent ∗ce)

  Read the trigger data of the central trigger in eventio format.
- int print_simtel_centralevent (IO_BUFFER ∗iobuf)

  Print the trigger data of the central trigger in eventio format.
- int write_simtel_trackevent (IO_BUFFER ∗iobuf, TrackEvent ∗tke)

  Write a tracking position in eventio format.
- int read_simtel_trackevent (IO_BUFFER ∗iobuf, TrackEvent ∗tke)

  Read a tracking position in eventio format.
- int print_simtel_trackevent (IO_BUFFER ∗iobuf)

  Print the tracking data in eventio format.
- int write_simtel_televt_head (IO_BUFFER ∗iobuf, TelEvent ∗te)

  Write the event header for data from one camera in eventio format.
- int read_simtel_televt_head (IO_BUFFER ∗iobuf, TelEvent ∗te)

  Read the event header for data from one camera in eventio format.
- int print_simtel_televt_head (IO_BUFFER ∗iobuf)

  Print the event header for data from one camera in eventio format.
- void **put_adcsum_as_uint16** (uint32_t ∗adc_sum, int n, IO_BUFFER ∗iobuf)
- void **get_adcsum_as_uint16** (uint32_t ∗adc_sum, int n, IO_BUFFER ∗iobuf)
- void **put_adcsum_differential** (uint32_t ∗adc_sum, int n, IO_BUFFER ∗iobuf)
- void **get_adcsum_differential** (uint32_t ∗adc_sum, int n, IO_BUFFER ∗iobuf)
- void **put_adcsample_differential** (uint16_t ∗adc_sample, int n, IO_BUFFER ∗iobuf)
- void **get_adcsample_differential** (uint16_t ∗adc_sample, int n, IO_BUFFER ∗iobuf)
- int write_simtel_teladc_sums (IO_BUFFER ∗iobuf, AdcData ∗raw)

  Write ADC sum data for one camera in eventio format.
- int read_simtel_teladc_sums (IO_BUFFER ∗iobuf, AdcData ∗raw)

  Write ADC sum data for one camera in eventio format.
- int print_simtel_teladc_sums (IO_BUFFER ∗iobuf)

  Print summed ADC data in eventio format.

- int write_simtel_teladc_samples (IO_BUFFER ∗iobuf, AdcData ∗raw)

  Write sampled ADC data in eventio format.
- int read_simtel_teladc_samples (IO_BUFFER ∗iobuf, AdcData ∗raw, int what)

  Read sampled ADC data in eventio format.
- int print_simtel_teladc_samples (IO_BUFFER ∗iobuf)

  Print sampled ADC data in eventio format.
- static void **adc_reset** (AdcData ∗raw)
- int write_simtel_aux_trace_digital (IO_BUFFER ∗iobuf, AuxTraceD ∗auxd)

  Write auxiliary digitized traces.
- int read_simtel_aux_trace_digital (IO_BUFFER ∗iobuf, AuxTraceD ∗auxd)

  Read auxiliary digitized traces.
- int print_simtel_aux_trace_digital (IO_BUFFER ∗iobuf)

  Print auxiliary digitized traces.
- int write_simtel_aux_trace_analog (IO_BUFFER ∗iobuf, AuxTraceA ∗auxa)

  Write auxiliary analog traces.
- int read_simtel_aux_trace_analog (IO_BUFFER ∗iobuf, AuxTraceA ∗auxa)

  Read auxiliary analog traces.
- int print_simtel_aux_trace_analog (IO_BUFFER ∗iobuf)

  Print auxiliary analog traces.
- int **write_simtel_pixeltrg_time** (IO_BUFFER ∗iobuf, PixelTrgTime ∗dt)
- int **read_simtel_pixeltrg_time** (IO_BUFFER ∗iobuf, PixelTrgTime ∗dt)
- int **print_simtel_pixeltrg_time** (IO_BUFFER ∗iobuf)
- static void build_list_for_hess_pixtime (PixelTiming ∗pixtm)

  A helper function finding the shorter of two possible formats for the list of pixels with any timing information.
- int write_simtel_pixtime (IO_BUFFER ∗iobuf, PixelTiming ∗pixtm)

  Write pixel timing parameters for selected pixels.
- int read_simtel_pixtime (IO_BUFFER ∗iobuf, PixelTiming ∗pixtm)

  Read pixel timing parameters for selected pixels.
- int print_simtel_pixtime (IO_BUFFER ∗iobuf)

  Print sampled ADC data in eventio format.
- int write_simtel_pixcalib (IO_BUFFER ∗iobuf, PixelCalibrated ∗pixcal)

  Write pixel intensities calibrated to (mean?) p.e.
- int read_simtel_pixcalib (IO_BUFFER ∗iobuf, PixelCalibrated ∗pixcal)

  Read pixel intensities calibrated to (mean?) p.e.
- int print_simtel_pixcalib (IO_BUFFER ∗iobuf)

  Print pixel intensities calibrated to (mean?) p.e.
- int write_simtel_telimage (IO_BUFFER ∗iobuf, ImgData ∗img, int what)

  Write image parameters for one telescope in eventio format.
- int read_simtel_telimage (IO_BUFFER ∗iobuf, ImgData ∗img)

  Read image parameters for one telescope in eventio format.
- int print_simtel_telimage (IO_BUFFER ∗iobuf)

  Print image parameters for one telescope in eventio format.
- int write_simtel_televent (IO_BUFFER ∗iobuf, TelEvent ∗te, int what)

  Write data for one telescope camera in eventio format.
- int read_simtel_televent (IO_BUFFER ∗iobuf, TelEvent ∗te, int what)

  Read data for one telescope camera in eventio format.
- int print_simtel_televent (IO_BUFFER ∗iobuf)

  Print data for one telescope camera in eventio format.
- int write_simtel_shower (IO_BUFFER ∗iobuf, ShowerParameters ∗sp)

  Write reconstructed shower parameters in eventio format.
- int read_simtel_shower (IO_BUFFER ∗iobuf, ShowerParameters ∗sp)

Read reconstructed shower parameters in eventio format.
- int print_simtel_shower (IO_BUFFER ∗iobuf)

  Print reconstructed shower parameters in eventio format.
- int write_simtel_event (IO_BUFFER ∗iobuf, FullEvent ∗ev, int what)

  Write the full array data of one event in eventio format.
- int read_simtel_event (IO_BUFFER ∗iobuf, FullEvent ∗ev, int what)

  Read the full array data of one event in eventio format.
- int print_simtel_event (IO_BUFFER ∗iobuf)

  Print the full array data of one event in eventio format.
- int write_simtel_calib_event (IO_BUFFER ∗iobuf, FullEvent ∗ev, int what, int type)

  Write a calibration event (pedestal, laser, led, ...) as an encapsulated raw data event.
- int read_simtel_calib_event (IO_BUFFER ∗iobuf, FullEvent ∗ev, int what, int ∗ptype)

  Read a calibration event (pedestal, laser, led, ...) as an encapsulated raw data event.
- int print_simtel_calib_event (IO_BUFFER ∗iobuf)

  Print a calibration event (pedestal, laser, led, ...) as an encapsulated raw data event.
- int read_simtel_calib_pe (IO_BUFFER ∗iobuf, MCEvent ∗mce, int ∗ptype)

  Read photo-electrons for a calibration event (pedestal, laser, led, ...) as an encapsulated raw data event.
- int print_simtel_calib_pe (IO_BUFFER ∗iobuf)

  Print a p.e.
- int write_simtel_mc_shower (IO_BUFFER ∗iobuf, MCShower ∗mcs)

  Write MC data for one simulated shower in eventio format.
- int read_simtel_mc_shower (IO_BUFFER ∗iobuf, MCShower ∗mcs)

  Read MC data for one simulated shower in eventio format.
- int print_simtel_mc_shower (IO_BUFFER ∗iobuf)

  Print MC data for one simulated shower in eventio format.
- int write_simtel_mc_event (IO_BUFFER ∗iobuf, MCEvent ∗mce)

  Write MC data for one use of a simulated shower in eventio format.
- int read_simtel_mc_event (IO_BUFFER ∗iobuf, MCEvent ∗mce)

  Read MC data for one use of a simulated shower in eventio format.
- int print_simtel_mc_event (IO_BUFFER ∗iobuf)

  Print MC data for one use of a simulated shower in eventio format.
- int write_simtel_mc_pe_sum (IO_BUFFER ∗iobuf, MCpeSum ∗mcpes)

  Write the numbers of photo-electrons detected from Cherenkov light in eventio format.
- int read_simtel_mc_pe_sum (IO_BUFFER ∗iobuf, MCpeSum ∗mcpes)

  Read the numbers of photo-electrons detected from Cherenkov light in eventio format.
- int print_simtel_mc_pe_sum (IO_BUFFER ∗iobuf)

  Print the numbers of photo-electrons detected from Cherenkov light in eventio format.
- int **write_simtel_mc_pixel_moni** (IO_BUFFER ∗iobuf, MCPixelMonitor ∗mcpixmon)
- int **read_simtel_mc_pixel_moni** (IO_BUFFER ∗iobuf, MCPixelMonitor ∗mcpixmon)
- int **print_simtel_mc_pixel_moni** (IO_BUFFER ∗iobuf)
- void **reset_htime** (HTime ∗t)
- void fill_htime_now (HTime ∗now)

  Fill the current time into a HTime structure.
- void copy_htime (HTime ∗t2, HTime ∗t1)

  Copy a time from one HTime structure into another one.
- int write_simtel_tel_monitor (IO_BUFFER ∗iobuf, TelMoniData ∗mon, int what)

  Write telescope camera monitoring information in eventio format.
- int read_simtel_tel_monitor (IO_BUFFER ∗iobuf, TelMoniData ∗mon)

  Read telescope camera monitoring information in eventio format.
- int print_simtel_tel_monitor (IO_BUFFER ∗iobuf)

  Print telescope camera monitoring information in eventio format.

- int write_simtel_laser_calib (IO_BUFFER ∗iobuf, LasCalData ∗lcd)

    Write a set of laser calibration data in eventio format.
- int read_simtel_laser_calib (IO_BUFFER ∗iobuf, LasCalData ∗lcd)

    Read a set of laser calibration data in eventio format.
- int print_simtel_laser_calib (IO_BUFFER ∗iobuf)

    Print a set of laser calibration data in eventio format.
- int write_simtel_run_stat (IO_BUFFER ∗iobuf, RunStat ∗rs)

    Write run statistics in eventio format.
- int read_simtel_run_stat (IO_BUFFER ∗iobuf, RunStat ∗rs)

    Read run statistics in eventio format.
- int print_simtel_run_stat (IO_BUFFER ∗iobuf)

    Print run statistics in eventio format.
- int write_simtel_mc_run_stat (IO_BUFFER ∗iobuf, MCRunStat ∗mcrs)

    Write Monte Carlo run statistics in eventio format.
- int read_simtel_mc_run_stat (IO_BUFFER ∗iobuf, MCRunStat ∗mcrs)

    Read Monte Carlo run statistics in eventio format.
- int print_simtel_mc_run_stat (IO_BUFFER ∗iobuf)

    Print Monte Carlo run statistics in eventio format.
- int read_simtel_mc_phot (IO_BUFFER ∗iobuf, MCEvent ∗mce)

    Read Monte Carlo photons and photo-electrons.
- int print_simtel_mc_phot (IO_BUFFER ∗iobuf)

    Print Monte Carlo photons and photo-electrons.
- int write_simtel_pixel_list (IO_BUFFER ∗iobuf, PixelList ∗pl, int telescope)

    Write lists of pixels (triggered, selected in image analysis, ...)
- int read_simtel_pixel_list (IO_BUFFER ∗iobuf, PixelList ∗pl, int ∗telescope)

    Read lists of pixels (triggered, selected in image analysis, ...)
- int print_simtel_pixel_list (IO_BUFFER ∗iobuf)

    Print lists of pixels (triggered, selected in image analysis, ...)

Variables

- static int hs_verbose = -1

    Should hessio print_...
- static int hs_maxprt = -1

    What is the maximum number of per pixel outputs?
- static int hs_dynamic = -1

    Should be check environment variables each time?
- static int **g_tel_idx** [3][H_MAX_TEL+1]
- static int **g_tel_idx_init** [3]
- static int **g_tel_idx_ref**

7.33.1 Detailed Description

Writing and reading of H.E.S.S.

/CTA data (or other simulation data produced by sim_telarray/sim_hessarray) in eventio format.

This file provides functions for writing and reading of H.E.S.S./CTA related data blocks or similar data for other telescope arrays. This software will attempt to be backward-compatible, i.e. to be able to read older data in slightly different formats - but we cannot guarantee that it really works. There is no attempt to write data in older formats. As always: use at your own risc.

Author

Konrad Bernlöhr

Date

July 2000 (initial version) to 2023

7.33.2 Function Documentation

7.33.2.1 find_tel_idx()

```
int find_tel_idx (
            int tel_id )
```

Lookup from telescope ID to offset number (index) in structures.

The lookup table must have been filled before with set_tel_idx(). When dealing with multiple lookups, use set_tel_idx_ref() first to select the lookup table to be used.

Parameters

tel_id	A telescope ID for which we want the index count.

Returns

>= 0 (index in the original list passed to set_tel_idx), -1 (not found in index, -2 (index not initialized).

7.33.2.2 print_simtel_aux_trace_analog()

```
int print_simtel_aux_trace_analog (
            IO_BUFFER * iobuf )
```

Print auxiliary analog traces.

< Must match the expected telescope ID when reading.

< Indicate what type of trace we have

< Time per auxiliary sample over time per normal FADC sample (typ.: 0.25)

< The number of traces coming from the camera.

< The length of each trace in FADC samples.

7.33.2.3 print_simtel_aux_trace_digital()

```
int print_simtel_aux_trace_digital (
             IO_BUFFER * iobuf )
```

Print auxiliary digitized traces.

< Must match the expected telescope ID when reading.

< Indicate what type of trace we have (1: DigitalSum trigger trace)

< Time per auxiliary sample over time per normal FADC sample (typ.: 1.0)

< The number of traces coming from the camera.

< The length of each trace in FADC samples.

7.33.2.4 print_simtel_calib_pe()

```
int print_simtel_calib_pe (
             IO_BUFFER * iobuf )
```

Print a p.e.

list for an internal calibration event from an encapsulated data block.

7.33.2.5 print_simtel_pixcalib()

```
int print_simtel_pixcalib (
             IO_BUFFER * iobuf )
```

Print pixel intensities calibrated to (mean?) p.e.

units.

7.33.2.6 read_simtel_pixcalib()

```
int read_simtel_pixcalib (
             IO_BUFFER * iobuf,
             PixelCalibrated * pixcal )
```

Read pixel intensities calibrated to (mean?) p.e.

units.

References simtel_pixel_calibrated_struct::known.

7.33.2.7 set_tel_idx()

```
void set_tel_idx (
             int ntel,
             int * idx )
```

Setup of telescope index lookup table.

Must be filled before first use of find_tel_idx() - which is automatically done when reading a run header data block. When dealing with multiple lookups, use set_tel_idx_ref() first to select the one to fill.

Parameters

ntel	The number of telescope following.
idx	The list of telescope IDs mapped to indices 0, 1, ...

7.33.2.8 set_tel_idx_ref()

```
void set_tel_idx_ref (
             int iref )
```

Switch between multiple telescope lookup tables.

Use this function when dealing simultaneously with multiple data streams for different array configurations. Both the set_tel_idx and the find_tel_idx will then work wit the selected choice of lookup table.

Parameters

iref	Which lookup table to use from now on (0<=iref<=2). Not switching lookup if iref is out of range.

Referenced by merge_data_from_io_block().

7.33.2.9 write_simtel_aux_trace_digital()

```
int write_simtel_aux_trace_digital (
             IO_BUFFER * iobuf,
             AuxTraceD * auxd )
```

Write auxiliary digitized traces.

There is no data reduction for auxiliary traces.

References simtel_aux_digital_trace::known, MAX_AUX_TRACE_D, simtel_aux_digital_trace::tel_id, simtel_aux↩_digital_trace::trace_data, and simtel_aux_digital_trace::trace_type.

7.33.2.10 write_simtel_event()

```
int write_simtel_event (
             IO_BUFFER * iobuf,
             FullEvent * ev,
             int what )
```

Write the full array data of one event in eventio format.

This can include raw data, tracking data, and central trigger data as gathered from the individual computers, as well as reconstructed parameters (image parameters, shower parameters).

7.33.2.11 write_simtel_laser_calib()

```
int write_simtel_laser_calib (
            IO_BUFFER * iobuf,
            LasCalData * lcd )
```

Write a set of laser calibration data in eventio format.

This may well change in a future revision (when more details are known how the real laser calibration should work).

7.33.2.12 write_simtel_mc_event()

```
int write_simtel_mc_event (
            IO_BUFFER * iobuf,
            MCEvent * mce )
```

Write MC data for one use of a simulated shower in eventio format.

This includes the core position shift with respect to the telescope array and the cross reference to the simulated shower.

7.33.2.13 write_simtel_mc_pe_sum()

```
int write_simtel_mc_pe_sum (
            IO_BUFFER * iobuf,
            MCpeSum * mcpes )
```

Write the numbers of photo-electrons detected from Cherenkov light in eventio format.

These are the 'true' numbers registered, not including photo-electrons from nightsky background.

7.33.2.14 write_simtel_mc_shower()

```
int write_simtel_mc_shower (
            IO_BUFFER * iobuf,
            MCShower * mcs )
```

Write MC data for one simulated shower in eventio format.

This includes data from the shower simulation itself, independent of how many times a shower is used and where the core position is shifted to with respect to the telescope array.

7.33.2.15 write_simtel_pixcalib()

```
int write_simtel_pixcalib (
            IO_BUFFER * iobuf,
            PixelCalibrated * pixcal )
```

Write pixel intensities calibrated to (mean?) p.e.

units.

References simtel_pixel_calibrated_struct::known.

7.33.2.16 write_simtel_run_stat()

```
int write_simtel_run_stat (
             IO_BUFFER * iobuf,
             RunStat * rs )
```

Write run statistics in eventio format.

This is pretty much dummy at this moment. Once we get closer to the real experiment, this data will certainly increase by a considerable amount.

7.33.2.17 write_simtel_shower()

```
int write_simtel_shower (
             IO_BUFFER * iobuf,
             ShowerParameters * sp )
```

Write reconstructed shower parameters in eventio format.

Note that the actual amount of data stored depends on what is actually available (as indicated in the 'result_bits').

7.33.2.18 write_simtel_tel_monitor()

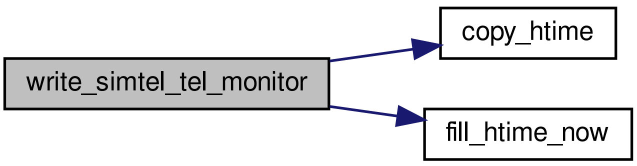
```
int write_simtel_tel_monitor (
             IO_BUFFER * iobuf,
             TelMoniData * mon,
             int what )
```

Write telescope camera monitoring information in eventio format.

What actually is written depends on the 'what' parameter. The general idea is to write only those things which have changed. Only when a target farm CPU becomes the target of the data stream, the full set of monitoring data is written.

References copy_htime(), fill_htime_now(), simtel_tel_monitor_struct::known, simtel_tel_monitor_struct::moni_↩
time, and simtel_tel_monitor_struct::new_parts.

Here is the call graph for this function:



7.33.2.19 write_simtel_teladc_samples()

```
int write_simtel_teladc_samples (
            IO_BUFFER * iobuf,
            AdcData * raw )
```

Write sampled ADC data in eventio format.

In contrast to sum data, no data reduction is applied so far. It is assumed that sampled data would be taken only for hardware tests, where the full information has to be maintained. If large amounts of sampled data are taken, a suitable data reduction method should be inserted here.

References simtel_tel_event_adc_struct::data_red_mode, and simtel_tel_event_adc_struct::zero_sup_mode.

7.33.2.20 write_simtel_teladc_sums()

```
int write_simtel_teladc_sums (
            IO_BUFFER * iobuf,
            AdcData * raw )
```

Write ADC sum data for one camera in eventio format.

The data can be optionally reduced (like writing only high-gain channels for pixels with low signals etc.) and zero-suppressed (not writing anything for pixels with very low signals).

References simtel_tel_event_adc_struct::data_red_mode, simtel_tel_event_adc_struct::known, simtel_tel_event↩
_adc_struct::list_known, and simtel_tel_event_adc_struct::zero_sup_mode.

7.33.2.21 write_simtel_televent()

```
int write_simtel_televent (
            IO_BUFFER * iobuf,
            TelEvent * te,
            int what )
```

Write data for one telescope camera in eventio format.

Depending on the 'what' parameter, either sampled or summed pixel values are expected to be in the 'te' structure. Writing of image paramaters is another option.

7.33.3 Variable Documentation

7.33.3.1 hs_verbose

```
int hs_verbose = -1  [static]
```

Should hessio print_...

functions be verbose?

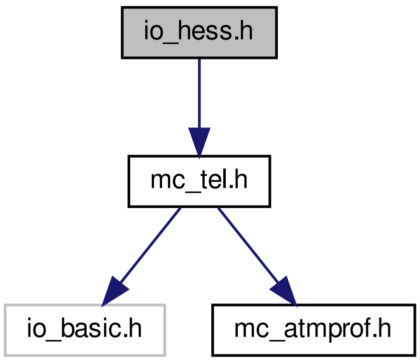
Referenced by hs_check_env(), and hs_reset_env().

7.34 io_hess.h File Reference

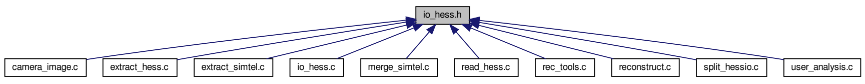
Definition and structures for H.E.S.S.

```
#include "mc_tel.h"
```

Include dependency graph for io_hess.h:



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



Data Structures

- struct simtel_run_header_struct

    Run header common to measured and simulated data.
- struct simtel_mc_run_header_struct

    MC run header.
- struct simtel_camera_settings_struct

Definition of camera optics settings.
- struct simtel_camera_organisation_struct

  Logical organisation of camera electronics channels.
- struct simtel_pixel_setting_struct

  Settings of pixel HV and thresholds.
- struct simtel_pixel_disabled_struct

  Pixels disabled in HV and/or trigger.
- struct simtel_camera_software_setting_struct

  Software settings used in camera process.
- struct simtel_tracking_setup_struct

  Definition of tracking parameters.
- struct simtel_pointing_correction_struct

  Pointing correction parameters.
- struct simtel_time_struct

  Breakdown of time into seconds since 1970.0 and nanoseconds.
- struct simtel_tel_event_adc_struct

  ADC data (either sampled or sum mode)
- struct simtel_aux_digital_trace

  Auxiliary digital trace (derived from FADC samples)
- struct simtel_aux_analog_trace

  Auxiliary analog trace (part of analog majority or sum trigger processing)
- struct simtel_pixel_timing_struct

  Time and amplitude values from a 'firmware'-like simple pulse analysis.
- struct simtel_pixeltrg_time_struct

  Times when pixels fired (not applicable for all trigger types).
- struct simtel_pixel_calibrated_struct

  Pixel signal intensities calibrated in some sort of p.e.
- struct simtel_pixel_list

  Lists of pixels (triggered, selected, etc.)
- struct simtel_tel_image_struct

  Image parameters.
- struct simtel_tel_event_data_struct

  Event raw and image data from one telescope.
- struct simtel_central_event_data_struct

  Central trigger event data.
- struct simtel_tracking_event_data_struct

  Tracking data interpolated for one event and one telescope.
- struct simtel_shower_parameter

  Reconstructed shower parameters.
- struct simtel_event_data_struct

  All data for one event.
- struct simtel_mc_shower_profile_struct

  Monte Carlo shower profile (sort of histogram).
- struct simtel_mc_shower_struct

  Shower specific data.
- struct simtel_mc_pe_sum_struct

  Sums of photo-electrons in MC (total and per pixel).
- struct simtel_mc_photons

  Collection of photons from Monte Carlo, as received from CORSIKA or LightEmission.
- struct simtel_fs_photon

  Single photon incident on focal surface, after ray-tracing in telescope optics.

- struct simtel_mc_fs_photons

    List of photons incident on focal surface.
- struct simtel_mc_pe_list

    Photo-electrons registered in pixels all listed individually.
- struct simtel_mc_pixel_monitor_struct

    Monte Carlo pixel 'monitoring' with parameters as actually used in simulation.
- struct simtel_mc_event_struct

    Monte Carlo event-specific data.
- struct simtel_tel_monitor_struct

    Monitoring data, traditionally emulating first-generation HESS cameras.
- struct simtel_laser_calib_data_struct

    Laser calibration data.
- struct simtel_run_end_statistics_struct

    End-of-run statistics.
- struct simtel_run_end_mc_statistics_struct

    MC end-of-run statistics.
- struct simtel_all_data_struct

    Container for all data.

Macros

- #define **IO_HESS_VERSION** 3
- #define HI_GAIN 0

    Which index refers to which type of channel:
- #define LO_GAIN 1

    Index to low-gain channels in adc_sum, adc_sample, pedestal, ...
- #define LARGE_TELESCOPE 1

    Maximum sizes for various arrays:
- #define **SMARTPIXEL** 1
- #define H_MAX_TEL 16

    Maximum number of telescopes handled.
- #define **H_MAX_TRG_PER_SECTOR** 1
- #define **H_MAX_PIX** 4095
- #define **H_MAX_SECTORS** (H_MAX_PIX∗H_MAX_TRG_PER_SECTOR)
- #define **H_MAX_DRAWERS** H_MAX_PIX
- #define H_MAX_GAINS 2

    Maximum number of different gains per PM.
- #define **H_MAX_PIXSECTORS** 4
- #define H_MAX_SLICES 128

    Maximum number of time slices handled.
- #define H_MAX_HOTPIX 5

    The max.
- #define H_MAX_PROFILE 10

    The max.
- #define **H_MAX_D_TEMP** 8
- #define **H_MAX_C_TEMP** 10
- #define H_MAX_FSHAPE 10000

    Max.
- #define **H_MAX_TRG_TYPES** 4
- #define H_CHECK_MAX()

    Compile-time override of the most relevant limits:

- #define RAWDATA_FLAG 0x01

  Flags used for saving and restoring event data:
- #define **RAWSUM_FLAG** 0x02
- #define **TRACKRAW_FLAG** 0x04
- #define **TRACKCOR_FLAG** 0x08
- #define **TRACKDATA_FLAG** (TRACKRAW_FLAG|TRACKCOR_FLAG)
- #define **IMG_BASE_FLAG** 0x10
- #define **IMG_ERR_FLAG** 0x20
- #define **IMG_34M_FLAG** 0x40
- #define **IMG_HOT_FLAG** 0x80
- #define **IMG_PIXTM_FLAG** 0x100
- #define **IMAGE_FLAG** (IMG_BASE_FLAG|IMG_ERR_FLAG|IMG_34M_FLAG|IMG_HOT_FLAG|IMG_↩ PIXTM_FLAG)
- #define **TIME_FLAG** 0x200
- #define **SHOWER_FLAG** 0x400
- #define **CALSUM_FLAG** 0x800
- #define IO_TYPE_SIMTEL_BASE 2000

  Never change the following numbers after MC data is created: (Now using the IO_TYPE_SIMTEL_...
- #define **IO_TYPE_SIMTEL_RUNHEADER** (IO_TYPE_SIMTEL_BASE+0)
- #define **IO_TYPE_SIMTEL_MCRUNHEADER** (IO_TYPE_SIMTEL_BASE+1)
- #define **IO_TYPE_SIMTEL_CAMSETTINGS** (IO_TYPE_SIMTEL_BASE+2)
- #define **IO_TYPE_SIMTEL_CAMORGAN** (IO_TYPE_SIMTEL_BASE+3)
- #define **IO_TYPE_SIMTEL_PIXELSET** (IO_TYPE_SIMTEL_BASE+4)
- #define **IO_TYPE_SIMTEL_PIXELDISABLE** (IO_TYPE_SIMTEL_BASE+5)
- #define **IO_TYPE_SIMTEL_CAMSOFTSET** (IO_TYPE_SIMTEL_BASE+6)
- #define **IO_TYPE_SIMTEL_POINTINGCOR** (IO_TYPE_SIMTEL_BASE+7)
- #define **IO_TYPE_SIMTEL_TRACKSET** (IO_TYPE_SIMTEL_BASE+8)
- #define **IO_TYPE_SIMTEL_CENTEVENT** (IO_TYPE_SIMTEL_BASE+9)
- #define **IO_TYPE_SIMTEL_TRACKEVENT** (IO_TYPE_SIMTEL_BASE+100)
- #define **IO_TYPE_SIMTEL_TELEVENT** (IO_TYPE_SIMTEL_BASE+200)
- #define **IO_TYPE_SIMTEL_EVENT** (IO_TYPE_SIMTEL_BASE+10)
- #define **IO_TYPE_SIMTEL_TELEVTHEAD** (IO_TYPE_SIMTEL_BASE+11)
- #define **IO_TYPE_SIMTEL_TELADCSUM** (IO_TYPE_SIMTEL_BASE+12)
- #define **IO_TYPE_SIMTEL_TELADCSAMP** (IO_TYPE_SIMTEL_BASE+13)
- #define **IO_TYPE_SIMTEL_TELIMAGE** (IO_TYPE_SIMTEL_BASE+14)
- #define **IO_TYPE_SIMTEL_SHOWER** (IO_TYPE_SIMTEL_BASE+15)
- #define **IO_TYPE_SIMTEL_PIXELTIMING** (IO_TYPE_SIMTEL_BASE+16)
- #define **IO_TYPE_SIMTEL_PIXELCALIB** (IO_TYPE_SIMTEL_BASE+17)
- #define **IO_TYPE_SIMTEL_MC_SHOWER** (IO_TYPE_SIMTEL_BASE+20)
- #define **IO_TYPE_SIMTEL_MC_EVENT** (IO_TYPE_SIMTEL_BASE+21)
- #define **IO_TYPE_SIMTEL_TEL_MONI** (IO_TYPE_SIMTEL_BASE+22)
- #define **IO_TYPE_SIMTEL_LASCAL** (IO_TYPE_SIMTEL_BASE+23)
- #define **IO_TYPE_SIMTEL_RUNSTAT** (IO_TYPE_SIMTEL_BASE+24)
- #define **IO_TYPE_SIMTEL_MC_RUNSTAT** (IO_TYPE_SIMTEL_BASE+25)
- #define **IO_TYPE_SIMTEL_MC_PE_SUM** (IO_TYPE_SIMTEL_BASE+26)
- #define **IO_TYPE_SIMTEL_PIXELLIST** (IO_TYPE_SIMTEL_BASE+27)
- #define **IO_TYPE_SIMTEL_CALIBEVENT** (IO_TYPE_SIMTEL_BASE+28)
- #define **IO_TYPE_SIMTEL_AUX_DIGITAL_TRACE** (IO_TYPE_SIMTEL_BASE+29)
- #define **IO_TYPE_SIMTEL_AUX_ANALOG_TRACE** (IO_TYPE_SIMTEL_BASE+30)
- #define **IO_TYPE_SIMTEL_FS_PHOT** (IO_TYPE_SIMTEL_BASE+31)
- #define **IO_TYPE_SIMTEL_PIXELTRG_TM** (IO_TYPE_SIMTEL_BASE+32)
- #define **IO_TYPE_SIMTEL_MC_PIXMON** (IO_TYPE_SIMTEL_BASE+33)
- #define **IO_TYPE_SIMTEL_CALIB_PE** (IO_TYPE_SIMTEL_BASE+34)
- #define IO_TYPE_HESS_BASE IO_TYPE_SIMTEL_BASE

The traditional definitions for these numbers all had HESS rather than SIMTEL in the name, and these will continue to work, for example in third-party code or with old sim_telarray code.

- #define **IO_TYPE_HESS_RUNHEADER** IO_TYPE_SIMTEL_RUNHEADER
- #define **IO_TYPE_HESS_MCRUNHEADER** IO_TYPE_SIMTEL_MCRUNHEADER
- #define **IO_TYPE_HESS_CAMSETTINGS** IO_TYPE_SIMTEL_CAMSETTINGS
- #define **IO_TYPE_HESS_CAMORGAN** IO_TYPE_SIMTEL_CAMORGAN
- #define **IO_TYPE_HESS_PIXELSET** IO_TYPE_SIMTEL_PIXELSET
- #define **IO_TYPE_HESS_PIXELDISABLE** IO_TYPE_SIMTEL_PIXELDISABLE
- #define **IO_TYPE_HESS_CAMSOFTSET** IO_TYPE_SIMTEL_CAMSOFTSET
- #define **IO_TYPE_HESS_POINTINGCOR** IO_TYPE_SIMTEL_POINTINGCOR
- #define **IO_TYPE_HESS_TRACKSET** IO_TYPE_SIMTEL_TRACKSET
- #define **IO_TYPE_HESS_CENTEVENT** IO_TYPE_SIMTEL_CENTEVENT
- #define **IO_TYPE_HESS_TRACKEVENT** IO_TYPE_SIMTEL_TRACKEVENT
- #define **IO_TYPE_HESS_TELEVENT** IO_TYPE_SIMTEL_TELEVENT
- #define **IO_TYPE_HESS_EVENT** IO_TYPE_SIMTEL_EVENT
- #define **IO_TYPE_HESS_TELEVTHEAD** IO_TYPE_SIMTEL_TELEVTHEAD
- #define **IO_TYPE_HESS_TELADCSUM** IO_TYPE_SIMTEL_TELADCSUM
- #define **IO_TYPE_HESS_TELADCSAMP** IO_TYPE_SIMTEL_TELADCSAMP
- #define **IO_TYPE_HESS_TELIMAGE** IO_TYPE_SIMTEL_TELIMAGE
- #define **IO_TYPE_HESS_SHOWER** IO_TYPE_SIMTEL_SHOWER
- #define **IO_TYPE_HESS_PIXELTIMING** IO_TYPE_SIMTEL_PIXELTIMING
- #define **IO_TYPE_HESS_PIXELCALIB** IO_TYPE_SIMTEL_PIXELCALIB
- #define **IO_TYPE_HESS_MC_SHOWER** IO_TYPE_SIMTEL_MC_SHOWER
- #define **IO_TYPE_HESS_MC_EVENT** IO_TYPE_SIMTEL_MC_EVENT
- #define **IO_TYPE_HESS_TEL_MONI** IO_TYPE_SIMTEL_TEL_MONI
- #define **IO_TYPE_HESS_LASCAL** IO_TYPE_SIMTEL_LASCAL
- #define **IO_TYPE_HESS_RUNSTAT** IO_TYPE_SIMTEL_RUNSTAT
- #define **IO_TYPE_HESS_MC_RUNSTAT** IO_TYPE_SIMTEL_MC_RUNSTAT
- #define **IO_TYPE_HESS_MC_PE_SUM** IO_TYPE_SIMTEL_MC_PE_SUM
- #define **IO_TYPE_HESS_PIXELLIST** IO_TYPE_SIMTEL_PIXELLIST
- #define **IO_TYPE_HESS_CALIBEVENT** IO_TYPE_SIMTEL_CALIBEVENT
- #define **IO_TYPE_HESS_AUX_DIGITAL_TRACE** IO_TYPE_SIMTEL_AUX_DIGITAL_TRACE
- #define **IO_TYPE_HESS_AUX_ANALOG_TRACE** IO_TYPE_SIMTEL_AUX_ANALOG_TRACE
- #define **IO_TYPE_HESS_FS_PHOT** IO_TYPE_SIMTEL_FS_PHOT
- #define **IO_TYPE_HESS_PIXELTRG_TM** IO_TYPE_SIMTEL_PIXELTRG_TM
- #define **IO_TYPE_HESS_MC_PIXMON** IO_TYPE_SIMTEL_MC_PIXMON
- #define **IO_TYPE_HESS_CALIB_PE** IO_TYPE_SIMTEL_CALIB_PE
- #define **HAS_CORSIKA_INTERACTION_DETAIL** 1
- #define MAX_AUX_TRACE_D 1

  Only one auxiliary digital trace.
- #define MAX_AUX_TRACE_A 4

  Up to four auxiliary analog traces.
- #define H_MAX_PIX_TIMES 7

  In addition to ADC we may (optionally) also have several types of timing data.
- #define PIX_TIME_PEAKPOS_TYPE 1

  Position of peak in time (slices since readout).
- #define PIX_TIME_STARTPOS_REL_TYPE 2

  Position of first rise above fraction of peak ampl.
- #define PIX_TIME_STARTPOS_ABS_TYPE 3

  Position of first rise above absolute threshold.
- #define PIX_TIME_WIDTH_REL_TYPE 4

  Width of pulse over fraction of peak ampl.
- #define PIX_TIME_WIDTH_ABS_TYPE 5

Width of pulse over absolute threshold (time over threshold).
- #define **hess_all_data_struct** simtel_all_data_struct
- #define **hess_tel_event_data_struct** simtel_tel_event_data_struct
- #define **hess_tel_monitor_struct** simtel_tel_monitor_struct
- #define **hess_tracking_event_data_struct** simtel_tracking_event_data_struct

Typedefs

- typedef struct simtel_run_header_struct RunHeader

  Use RunHeader rather than the plain struct name in any code.
- typedef struct simtel_mc_run_header_struct MCRunHeader

  Use MCRunHeader rather than the plain struct name in any code.
- typedef struct simtel_camera_settings_struct CameraSettings

  Use CameraSettings rather than the plain struct name in any code.
- typedef struct simtel_camera_organisation_struct CameraOrganisation

  Use CameraOrganisation rather than the plain struct name in any code.
- typedef struct simtel_pixel_setting_struct PixelSetting

  Use PixelSetting rather than the plain struct name in any code.
- typedef struct simtel_pixel_disabled_struct PixelDisabled

  Use PixelDisabled rather than the plain struct name in any code.
- typedef struct simtel_camera_software_setting_struct CameraSoftSet

  Use CameraSoftSet rather than the plain struct name in any code.
- typedef struct simtel_tracking_setup_struct TrackingSetup

  Use TrackingSetup rather than the plain struct name in any code.
- typedef struct simtel_pointing_correction_struct PointingCorrection

  Use PointingCorrection rather than the plain struct name in any code.
- typedef struct simtel_time_struct HTime

  Use HTime rather than the plain struct name in any code.
- typedef struct simtel_tel_event_adc_struct AdcData

  Use AdcData rather than the plain struct name in any code.
- typedef struct simtel_aux_digital_trace AuxTraceD

  Use AuxTraceD rather than the plain struct name in any code.
- typedef struct simtel_aux_analog_trace AuxTraceA

  Use AuxTraceA rather than the plain struct name in any code.
- typedef struct simtel_pixel_timing_struct PixelTiming

  Use PixelTiming rather than the plain struct name in any code.
- typedef struct simtel_pixeltrg_time_struct PixelTrgTime

  Use PixelTrgTime rather than the plain struct name in any code.
- typedef struct simtel_pixel_calibrated_struct PixelCalibrated

  Use PixelCalibrated rather than the plain struct name in any code.
- typedef struct simtel_pixel_list PixelList

  Use PixelList rather than the plain struct name in any code.
- typedef struct simtel_tel_image_struct ImgData

  Use ImgData rather than the plain struct name in any code.
- typedef struct simtel_tel_event_data_struct TelEvent

  Use TelEvent rather than the plain struct name in any code.
- typedef struct simtel_central_event_data_struct CentralEvent

  Use CentralEvent rather than the plain struct name in any code.
- typedef struct simtel_tracking_event_data_struct TrackEvent

  Use TrackEvent rather than the plain struct name in any code.

- typedef struct simtel_shower_parameter ShowerParameters

    Use ShowerParameters rather than the plain struct name in any code.
- typedef struct simtel_event_data_struct FullEvent

    Use FullEvent rather than the plain struct name in any code.
- typedef struct simtel_mc_shower_profile_struct ShowerProfile

    Use ShowerProfile rather than the plain struct name in any code.
- typedef struct simtel_mc_shower_struct MCShower

    Use MCShower rather than the plain struct name in any code.
- typedef struct simtel_mc_pe_sum_struct MCpeSum

    Use MCpeSum rather than the plain struct name in any code.
- typedef struct simtel_mc_photons MCphotons

    Use MCphotons rather than the plain struct name in any code.
- typedef struct simtel_fs_photon FSphoton

    Use FSphoton rather than the plain struct name in any code.
- typedef struct simtel_mc_fs_photons MCfsPhotons

    Use MCfsPhotons rather than the plain struct name in any code.
- typedef struct simtel_mc_pe_list MCpeList

    Use MCpeList rather than the plain struct name in any code.
- typedef struct simtel_mc_pixel_monitor_struct MCPixelMonitor

    Use MCPixelMonitor rather than the plain struct name in any code.
- typedef struct simtel_mc_event_struct MCEvent

    Use MCEvent rather than the plain struct name in any code.
- typedef struct simtel_tel_monitor_struct TelMoniData

    Use TelMoniData rather than the plain struct name in any code.
- typedef struct simtel_laser_calib_data_struct LasCalData

    Use LasCalData rather than the plain struct name in any code.
- typedef struct simtel_run_end_statistics_struct RunStat

    Use RunStat rather than the plain struct name in any code.
- typedef struct simtel_run_end_mc_statistics_struct MCRunStat

    Use MCRunStat rather than the plain struct name in any code.
- typedef struct simtel_all_data_struct AllHessData

    Use AllHessData rather than the plain struct name in any code.

Functions

- void check_hessio_max (int ncheck, int max_tel, int max_pix, int max_sectors, int max_drawers, int max_↩ pixsectors, int max_slices, int max_hotpix, int max_profile, int max_d_temp, int max_c_temp, int max_gains)

    Support for checking if user functions are compiled with the same limits as the library.
- void **show_hessio_max** (void)

7.34.1 Detailed Description

Definition and structures for H.E.S.S.

/CTA data in eventio format.

This file contains definitions and data structures used originally for writing and reading HESS data (both Monte Carlo and real data) in the eventio format. For sim_telarray output, this is the native data format. Beyond the original needs for HESS, it has seen many extensions for CTA simulation data and other instruments.

Author

Konrad Bernlöhr

Date

initial version: July 2000

2000 to 2023

7.34.2 Macro Definition Documentation

7.34.2.1 H_CHECK_MAX

```
#define H_CHECK_MAX( )
```

Value:

```
check_hessio_max(11,H_MAX_TEL,H_MAX_PIX,H_MAX_SECTORS,\
H_MAX_DRAWERS,H_MAX_PIXSECTORS,H_MAX_SLICES,H_MAX_HOTPIX,H_MAX_PROFILE,\
H_MAX_D_TEMP,H_MAX_C_TEMP,H_MAX_GAINS);
```

Compile-time override of the most relevant limits:

Macro expanding into a function call checking if user function is taking the same maximum array sizes as the library.

7.34.2.2 H_MAX_FSHAPE

```
#define H_MAX_FSHAPE 10000
```

Max.

number of (sub-) samples of reference pulse shapes.

7.34.2.3 H_MAX_HOTPIX

```
#define H_MAX_HOTPIX 5
```

The max.

size of the list of hottest pix.

7.34.2.4 H_MAX_PROFILE

```
#define H_MAX_PROFILE 10
```

The max.

number of MC shower profiles.

7.34.2.5 HI_GAIN

```
#define HI_GAIN 0
```

Which index refers to which type of channel:

Index to high-gain channels in adc_sum, adc_sample, pedestal, ...

7.34.2.6 IO_TYPE_SIMTEL_BASE

```
#define IO_TYPE_SIMTEL_BASE 2000
```

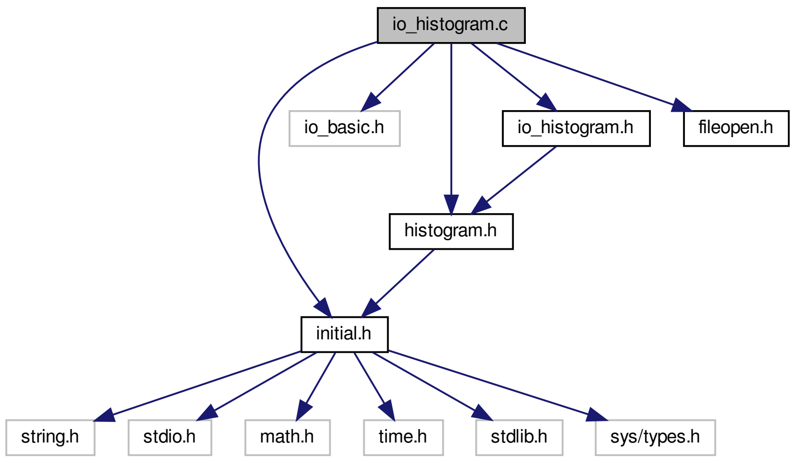
Never change the following numbers after MC data is created: (Now using the IO_TYPE_SIMTEL_...

as the primary definition but the traditional IO_TYPE_HESS_... will remain to work.)

7.35 io_histogram.c File Reference

This file implements I/O for 1-D and 2-D histograms.

```
#include "initial.h"
#include "io_basic.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
```
Include dependency graph for io_histogram.c:



Macros

- #define **__attribute__**(a) /∗ Ignore gcc specials with other compilers ∗/

Functions

- int write_all_histograms (const char ∗fname)

  Save all available histograms into the file with the given name.
- int **read_histogram_file** (const char ∗fname, int add_flag)
- int **read_histogram_file_x** (const char ∗fname, int add_flag, const long ∗xcld_ids, int nxcld)
- int write_histograms (HISTOGRAM ∗∗phisto, int nhisto, IO_BUFFER ∗iobuf)

  Save specific histograms or all allocated histograms.
- int read_histograms (HISTOGRAM ∗∗phisto, int nhisto, IO_BUFFER ∗iobuf)

  Read and allocate histograms and optionally return histogram pointers to caller.
- int read_histograms_x (HISTOGRAM ∗∗phisto, int nhisto, const long ∗xcld_ids, int nxcld, IO_BUFFER ∗iobuf)

  Read and allocate histograms and optionally return histogram pointers to caller.
- int print_histograms (IO_BUFFER ∗iobuf)

  Print out some basics about histogram data as we read it.

7.35.1 Detailed Description

This file implements I/O for 1-D and 2-D histograms.

Author

Konrad Bernloehr

Date

1993 to 2021

7.35.2 Function Documentation

7.35.2.1 print_histograms()

```
int print_histograms (
             IO_BUFFER ∗ iobuf )
```

Print out some basics about histogram data as we read it.

Parameters

iobuf	The input iobuf descriptor.

Returns

>= 0 (O.k., no. of histograms read), -1 (error), -2 (e.o.d.)

7.35.2.2 read_histograms()

```
int read_histograms (
            HISTOGRAM ** phisto,
            int nhisto,
            IO_BUFFER * iobuf )
```

Read and allocate histograms and optionally return histogram pointers to caller.

Parameters

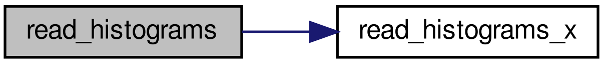
phisto	Pointer to vector of histogram pointers or NULL.
nhisto	The no. of elements in the phisto vector, i.e. the max. no. of histograms of which the histogram pointer can be returned to the caller. If negative, histograms contents are added to existing histograms of the same ID.
iobuf	The input iobuf descriptor.

Returns

>= 0 (O.k., no. of histograms read), -1 (error), -2 (e.o.d.)

References read_histograms_x().

Here is the call graph for this function:



7.35.2.3 read_histograms_x()

```
int read_histograms_x (
            HISTOGRAM ** phisto,
            int nhisto,
            const long * xcld_ids,
            int nxcld,
            IO_BUFFER * iobuf )
```

Read and allocate histograms and optionally return histogram pointers to caller.

This extended version allows to exclude a list of histogram IDs from being kept or added.

Parameters

phisto	Pointer to vector of histogram pointers or NULL.
nhisto	The no. of elements in the phisto vector, i.e. the max. no. of histograms of which the histogram pointer can be returned to the caller. If negative, histograms contents are added to existing histograms of the same ID.
xcld_ids	Pointer to vector of histogram IDs to be excluded.
ncxld	Number of histogram IDs to be excluded.
iobuf	The input iobuf descriptor.

Returns

$>=$ 0 (O.k., no. of histograms read), -1 (error), -2 (e.o.d.)

Referenced by read_histograms().

7.35.2.4 write_histograms()

```
int write_histograms (
            HISTOGRAM ** phisto,
            int nhisto,
            IO_BUFFER * iobuf )
```

Save specific histograms or all allocated histograms.

Parameters

phisto	Pointer to vector of histogram pointers or NULL.
nhisto	The no. of histograms to be saved or -1. If phisto==NULL and nhisto==-1 then all allocated histograms (in the linked list of histograms) are saved.
iobuf	The output iobuf descriptor.

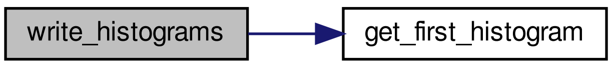
Returns

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

References get_first_histogram(), histogram::ident, and histogram::next.

Referenced by write_all_histograms(), and write_dst_histos().

Here is the call graph for this function:

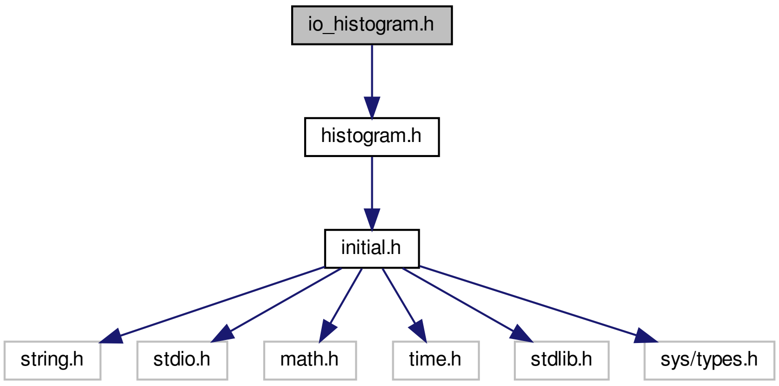


7.36 io_histogram.h File Reference

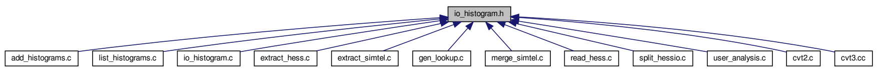
Declarations for eventio I/O of histograms.

```
#include "histogram.h"
```

Include dependency graph for io_histogram.h:



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



Functions

- int write_histograms (HISTOGRAM ∗∗phisto, int nhisto, IO_BUFFER ∗iobuf)

  Save specific histograms or all allocated histograms.
- int read_histograms (HISTOGRAM ∗∗phisto, int nhisto, IO_BUFFER ∗iobuf)

  Read and allocate histograms and optionally return histogram pointers to caller.
- int read_histograms_x (HISTOGRAM ∗∗phisto, int nhisto, const long ∗xcld_ids, int nxcld, IO_BUFFER ∗iobuf)

  Read and allocate histograms and optionally return histogram pointers to caller.
- int print_histograms (IO_BUFFER ∗iobuf)

  Print out some basics about histogram data as we read it.
- int write_all_histograms (const char ∗fname)

  Save all available histograms into the file with the given name.
- int **read_histogram_file** (const char ∗fname, int add_flag)
- int **read_histogram_file_x** (const char ∗fname, int add_flag, const long ∗xcld_ids, int nxcld)

7.36.1 Detailed Description

Declarations for eventio I/O of histograms.

Date

1993 ro 2023

Author

Konrad Bernloehr

7.36.2 Function Documentation

7.36.2.1 print_histograms()

```
int print_histograms (
             IO_BUFFER * iobuf )
```

Print out some basics about histogram data as we read it.

Parameters

iobuf	The input iobuf descriptor.

Returns

>= 0 (O.k., no. of histograms read), -1 (error), -2 (e.o.d.)

7.36.2.2 read_histograms()

```
int read_histograms (
             HISTOGRAM ** phisto,
             int nhisto,
             IO_BUFFER * iobuf )
```

Read and allocate histograms and optionally return histogram pointers to caller.

Parameters

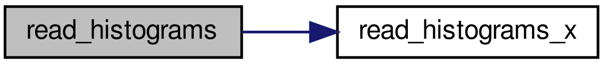
phisto	Pointer to vector of histogram pointers or NULL.
nhisto	The no. of elements in the phisto vector, i.e. the max. no. of histograms of which the histogram pointer can be returned to the caller. If negative, histograms contents are added to existing histograms of the same ID.
iobuf	The input iobuf descriptor.

Returns

>= 0 (O.k., no. of histograms read), -1 (error), -2 (e.o.d.)

References read_histograms_x().

Here is the call graph for this function:



7.36.2.3 read_histograms_x()

```
int read_histograms_x (
             HISTOGRAM ** phisto,
             int nhisto,
             const long * xcld_ids,
             int nxcld,
             IO_BUFFER * iobuf )
```

Read and allocate histograms and optionally return histogram pointers to caller.

This extended version allows to exclude a list of histogram IDs from being kept or added.

Parameters

phisto	Pointer to vector of histogram pointers or NULL.
nhisto	The no. of elements in the phisto vector, i.e. the max. no. of histograms of which the histogram pointer can be returned to the caller. If negative, histograms contents are added to existing histograms of the same ID.
xcld_ids	Pointer to vector of histogram IDs to be excluded.
ncxld	Number of histogram IDs to be excluded.
iobuf	The input iobuf descriptor.

Returns

>= 0 (O.k., no. of histograms read), -1 (error), -2 (e.o.d.)

Referenced by read_histograms().

7.36.2.4 write_histograms()

```
int write_histograms (
             HISTOGRAM ** phisto,
             int nhisto,
             IO_BUFFER * iobuf )
```

Save specific histograms or all allocated histograms.

Parameters

phisto	Pointer to vector of histogram pointers or NULL.
nhisto	The no. of histograms to be saved or -1. If phisto==NULL and nhisto==-1 then all allocated histograms (in the linked list of histograms) are saved.
iobuf	The output iobuf descriptor.

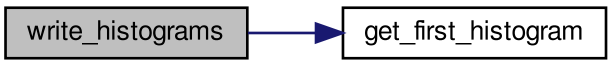
Returns

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

References get_first_histogram(), histogram::ident, and histogram::next.

Referenced by write_all_histograms(), and write_dst_histos().

Here is the call graph for this function:



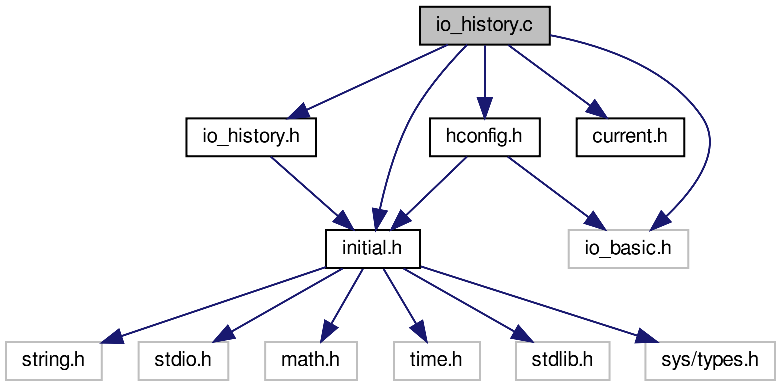
7.37 io_history.c File Reference

Record history of configuration settings/commands.

```
#include "initial.h"
#include "io_basic.h"
#include "io_history.h"
#include "current.h"
```

```
#include "hconfig.h"
```

Include dependency graph for io_history.c:



Functions

- static void **listtime** (time_t t, FILE ∗f)
- int push_command_history (int argc, char ∗∗argv)

  Save the command line for later output in a history block.
- int push_config_history (const char ∗line, int noreplace)

  Save a line of configuration text for later output in a history block.
- int clear_hstruct (HSTRUCT ∗h)

  Clear and free all elements along one linked list of history elements.
- int clear_histcont (HistoryContainer ∗hc)

  Clear and free all linked list of history elements in a history container.
- int write_history (long id, IO_BUFFER ∗iobuf)

  Write the block of accumulated history lines (command line and configuration lines) to an I/O buffer.
- int read_history (IO_BUFFER ∗iobuf, HistoryContainer ∗hc)

  Read the block of accumulated history lines (command line and configuration lines) from an I/O buffer and try to split the configuration history by telescope, if the transition can be found.
- int write_config_history (const char ∗htext, long htime, long id, IO_BUFFER ∗iobuf)

  Write a configuration history line to an I/O buffer.
- int list_history (IO_BUFFER ∗iobuf, FILE ∗file)

  List history block contents on standard output or other file.
- int print_history (IO_BUFFER ∗iobuf)

  Stub for list_history(), to stdout only.
- void **set_metaparam_id** (MetaParamList ∗lst, long id)
- int fill_metaparam (MetaParamList ∗lst, const char ∗∗names, const char ∗∗values, size_t npar, long id)

  Fill in meta parameters to the linked list after (re-)initialising it.
- int add_metaparam (MetaParamList ∗lst, const char ∗name, const char ∗value)

  Add a meta parameter to the linked list or change a matching enty.
- int clear_metaparam (MetaParamList ∗lst)

  Clear a list of meta parameters.
- int write_metaparam (IO_BUFFER ∗iobuf, const MetaParamList ∗lst)

  Write a data block of meta parameters.

- int read_metaparam (IO_BUFFER ∗iobuf, MetaParamList ∗lst)

  Read from a data block of meta parameters.
- int print_metaparam (IO_BUFFER ∗iobuf)

  Display the contents of a data block of meta parameters.
- int show_metaparam (const MetaParamList ∗lst)

  Display the contents in a linked list of meta parameters, independent of its EventIO representation.
- const char ∗ search_metaparam (const MetaParamList ∗lst, const char ∗name)

  Search for a specific metaparameter by name (case-insensitive, first match returned) in a linked list of meta parameters.

Variables

- static char ∗ cmdline = NULL

  A copy of the program's command line.
- static time_t cmdtime

  The time when the program was started.
- static HSTRUCT ∗ configs = NULL

  Start of configuration history.

7.37.1 Detailed Description

Record history of configuration settings/commands.

This code has not been adapted for multi-threading.

Author

Konrad Bernloehr

Date

1997 to 2022

7.37.2 Function Documentation

7.37.2.1 add_metaparam()

```
int add_metaparam (
            MetaParamList ∗ lst,
            const char ∗ name,
            const char ∗ value )
```

Add a meta parameter to the linked list or change a matching enty.

Parameters

lst	The starting point of a linked list. Must exist but can be without items.
name	Name of the item to add. Must exist and be non-empty. @paran value Value of the item to add. Must exist.

Returns

0 (OK), -1 or other non-zero (problem)

References meta_param_list::first, meta_param_item::name, meta_param_item::next, and meta_param_item↩ ::value.

7.37.2.2 clear_histcont()

```
int clear_histcont (
            HistoryContainer * hc )
```

Clear and free all linked list of history elements in a history container.

The container itself will only contain null pointers afterwards but no attempts are made to free() it as well - it could just as well come from the stack.

Parameters

hc	Pointer to history container.

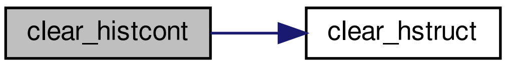
Returns

Number of elements released.

References history_container_struct::cfg_global, history_container_struct::cfg_tel, clear_hstruct(), history_↩ container_struct::cmdline, history_container_struct::id, and history_container_struct::ntel.

Referenced by read_history().

Here is the call graph for this function:



7.37.2.3 clear_hstruct()

```
int clear_hstruct (
            HSTRUCT * h )
```

Clear and free all elements along one linked list of history elements.

Parameters

h	Start of linked list of HSTRUCTs.

Returns

Number of elements released.

References history_struct::next, and history_struct::time.

Referenced by clear_histcont().

7.37.2.4 clear_metaparam()

```
int clear_metaparam (
            MetaParamList * lst )
```

Clear a list of meta parameters.

All pointers along the linked list must be dynamically allocated.

Parameters

lst	The starting point of a linked list. Must exist.

Returns

0 (OK), -1 or other non-zero (problem)

References meta_param_list::first, meta_param_list::ident, meta_param_item::name, meta_param_item::next, and meta_param_item::value.

Referenced by fill_metaparam().

7.37.2.5 fill_metaparam()

```
int fill_metaparam (
            MetaParamList * lst,
            const char ** names,
            const char ** values,
            size_t npar,
            long id )
```

Fill in meta parameters to the linked list after (re-)initialising it.

Parameters

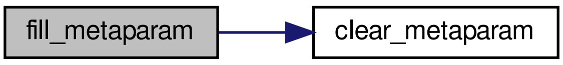
lst	The starting point of a linked list. Must exist but can be without items.
names	Array of names of the items to add. Array and all names must exist and be non-empty. @paran values Array of values of the items to add. Array and and values must exist (but can be empty).

Returns

0 (OK), -1 or other non-zero (problem)

References clear_metaparam(), meta_param_list::first, meta_param_list::ident, meta_param_item::name, meta_↩ param_item::next, and meta_param_item::value.

Here is the call graph for this function:



7.37.2.6 list_history()

```
int list_history (
            IO_BUFFER * iobuf,
            FILE * file )
```

List history block contents on standard output or other file.

Parameters

iobuf	I/O buffer descriptor
file	Optional open output stream (NULL -> stdout).

Returns

0 (o.k.), <0 (error with I/O buffer)

Referenced by print_history().

7.37.2.7 print_history()

```
int print_history (
            IO_BUFFER * iobuf )
```

Stub for list_history(), to stdout only.

Parameters

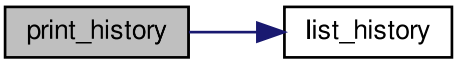
iobuf	I/O buffer descriptor

Returns

0 (o.k.), <0 (error with I/O buffer)

References list_history().

Here is the call graph for this function:



7.37.2.8 print_metaparam()

```
int print_metaparam (
            IO_BUFFER * iobuf )
```

Display the contents of a data block of meta parameters.

Parameters

iobuf	I/O buffer descriptor

Returns

0 (OK), -1 or other non-zero (problem)

7.37.2.9 push_command_history()

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

Save the command line for later output in a history block.

Parameters

argc	Number of command line arguments (incl. command)
argv	Pointers to argument text strings

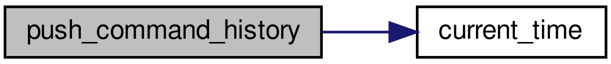
Returns

0 (o.k.), -1 (invalid argument or no memory)

References cmdline, cmdtime, and current_time().

Referenced by main().

Here is the call graph for this function:



7.37.2.10 push_config_history()

```
int push_config_history (
            const char * line,
            int noreplace )
```

Save a line of configuration text for later output in a history block.

If any configuration text for the same keyword was present before, it may be replaced by the new text, even if the new setting does not replace all old settings.

Parameters

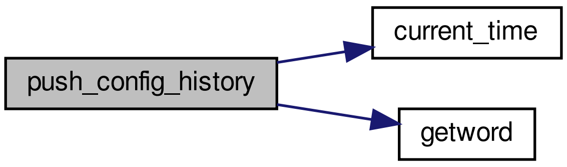
line	Configuration text line.
replace	Replace old text for same keyword (1) or not (0).

Returns

0 (o.k.), -1 (memory allocation failed)

References configs, current_time(), getword(), history_struct::next, and history_struct::time.

Here is the call graph for this function:



7.37.2.11 read_history()

```
int read_history (
            IO_BUFFER * iobuf,
            HistoryContainer * hc )
```

Read the block of accumulated history lines (command line and configuration lines) from an I/O buffer and try to split the configuration history by telescope, if the transition can be found.

Note that in extracted/split/merged sim_telarray data the correspondence to current telescope IDs is lost. For other programs, no telescope transition is recognized, i.e. all configuration is global.

Parameters

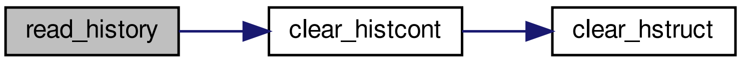
iobuf	I/O buffer descriptor
hc	Pointer to history container.

Returns

0 (o.k.), <0 (error with I/O buffer)

References clear_histcont().

Here is the call graph for this function:



7.37.2.12 read_metaparam()

```
int read_metaparam (
            IO_BUFFER * iobuf,
            MetaParamList * lst )
```

Read from a data block of meta parameters.

Parameters

iobuf	I/O buffer descriptor
lst	The starting point of a linked list. Must exist.

Returns

0 (OK), -1 or other non-zero (problem)

7.37.2.13 search_metaparam()

```
const char* search_metaparam (
            const MetaParamList * lst,
            const char * name )
```

Search for a specific metaparameter by name (case-insensitive, first match returned) in a linked list of meta parameters.

NULL is returned if there is no match.

Since metaparameters are stored only once per run (and telescope), and there should not be many of them, a linear search should be good enough.

Parameters

lst	The starting point of a linked list. Must exist but can be without items.
name	The name of the metaparameter searched for.

Returns

Value of metaparameter (first match found), or NULL pointer.

References meta_param_list::first, meta_param_item::name, meta_param_item::next, and meta_param_item↩ ::value.

7.37.2.14 show_metaparam()

```
int show_metaparam (
            const MetaParamList * lst )
```

Display the contents in a linked list of meta parameters, independent of its EventIO representation.

Parameters

lst	The starting point of a linked list. Must exist but can be without items.

Returns

0 (OK), -1 or other non-zero (problem)

References meta_param_list::first, meta_param_list::ident, meta_param_item::name, meta_param_item::next, and meta_param_item::value.

7.37.2.15 write_config_history()

```
int write_config_history (
            const char * htext,
            long htime,
            long id,
            IO_BUFFER * iobuf )
```

Write a configuration history line to an I/O buffer.

Parameters

htext	Text of configuration line
htime	Time when the configuration was set.
id	Identifier (detector number)
iobuf	I/O buffer descriptor

Returns

0 (o.k.), <0 (error with I/O buffer)

7.37.2.16 write_history()

```
int write_history (
            long id,
            IO_BUFFER * iobuf )
```

Write the block of accumulated history lines (command line and configuration lines) to an I/O buffer.

Parameters

id	Identifier (detector number)
iobuf	I/O buffer descriptor

Returns

0 (o.k.), <0 (error with I/O buffer)

7.37.2.17 write_metaparam()

```
int write_metaparam (
            IO_BUFFER * iobuf,
            const MetaParamList * lst )
```

Write a data block of meta parameters.

Parameters

iobuf	I/O buffer descriptor
lst	The starting point of a linked list. Must exist.

Returns

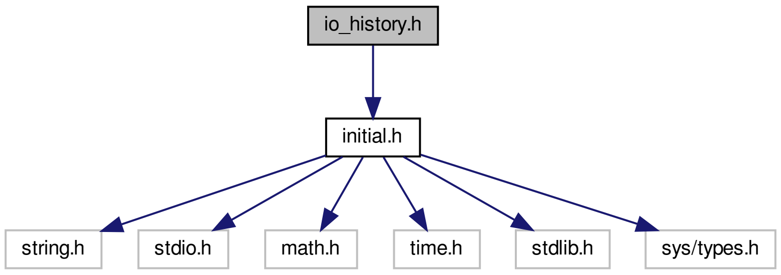
0 (OK), -1 or other non-zero (problem)

7.38 io_history.h File Reference

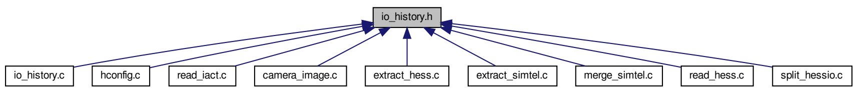
Record history of configuration settings/commands.

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

Include dependency graph for io_history.h:



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



Data Structures

- struct history_struct

  Use to build a linked list of configuration history.
- struct history_container_struct
- struct meta_param_item

  A history meta parameter item consists of a parameter name and its text-formatted value, both as text strings.
- struct meta_param_list

  The linked MetaParamItem list for one ID (-1=global, detector ID otherwise) are registered under one list starting point.

Macros

- #define **IO_TYPE_HISTORY** 70
- #define **IO_TYPE_CMD_HIST** 71
- #define **IO_TYPE_CFG_HIST** 72
- #define **IO_TYPE_METAPARAM** 75
- #define **WITH_METAPARAM_HISTORY** 1

Typedefs

- typedef struct history_struct **HSTRUCT**
- typedef struct history_container_struct **HistoryContainer**
- typedef struct meta_param_item **MetaParamItem**
- typedef struct meta_param_list **MetaParamList**

Functions

- int clear_hstruct (HSTRUCT ∗h)

  Clear and free all elements along one linked list of history elements.
- int clear_histcont (HistoryContainer ∗hc)

  Clear and free all linked list of history elements in a history container.
- int push_command_history (int argc, char ∗∗argv)

  Save the command line for later output in a history block.
- int push_config_history (const char ∗line, int noreplace)

  Save a line of configuration text for later output in a history block.
- int write_history (long id, IO_BUFFER ∗iobuf)

  Write the block of accumulated history lines (command line and configuration lines) to an I/O buffer.
- int read_history (IO_BUFFER ∗iobuf, HistoryContainer ∗hc)

  Read the block of accumulated history lines (command line and configuration lines) from an I/O buffer and try to split the configuration history by telescope, if the transition can be found.
- int write_config_history (const char ∗htext, long htime, long id, IO_BUFFER ∗iobuf)

  Write a configuration history line to an I/O buffer.
- int list_history (IO_BUFFER ∗iobuf, FILE ∗file)

  List history block contents on standard output or other file.
- int print_history (IO_BUFFER ∗iobuf)

  Stub for list_history(), to stdout only.
- void **set_metaparam_id** (MetaParamList ∗lst, long id)
- int fill_metaparam (MetaParamList ∗lst, const char ∗∗names, const char ∗∗values, size_t npar, long id)

  Fill in meta parameters to the linked list after (re-)initialising it.
- int add_metaparam (MetaParamList ∗lst, const char ∗name, const char ∗value)

  Add a meta parameter to the linked list or change a matching enty.
- int clear_metaparam (MetaParamList ∗lst)

  Clear a list of meta parameters.
- int write_metaparam (IO_BUFFER ∗iobuf, const MetaParamList ∗lst)

  Write a data block of meta parameters.
- int read_metaparam (IO_BUFFER ∗iobuf, MetaParamList ∗lst)

  Read from a data block of meta parameters.
- int print_metaparam (IO_BUFFER ∗iobuf)

  Display the contents of a data block of meta parameters.
- int show_metaparam (const MetaParamList ∗lst)

  Display the contents in a linked list of meta parameters, independent of its EventIO representation.
- const char ∗ search_metaparam (const MetaParamList ∗lst, const char ∗name)

  Search for a specific metaparameter by name (case-insensitive, first match returned) in a linked list of meta parameters.

7.38.1 Detailed Description

Record history of configuration settings/commands.

Author

Konrad Bernloehr

Date

1997 to 2021

7.38.2 Function Documentation

7.38.2.1 add_metaparam()

```
int add_metaparam (
            MetaParamList * lst,
            const char * name,
            const char * value )
```

Add a meta parameter to the linked list or change a matching enty.

Parameters

lst	The starting point of a linked list. Must exist but can be without items.
name	Name of the item to add. Must exist and be non-empty. @paran value Value of the item to add. Must exist.

Returns

0 (OK), -1 or other non-zero (problem)

References meta_param_list::first, meta_param_item::name, meta_param_item::next, and meta_param_item↩ ::value.

7.38.2.2 clear_histcont()

```
int clear_histcont (
            HistoryContainer * hc )
```

Clear and free all linked list of history elements in a history container.

The container itself will only contain null pointers afterwards but no attempts are made to free() it as well - it could just as well come from the stack.

Parameters

hc	Pointer to history container.

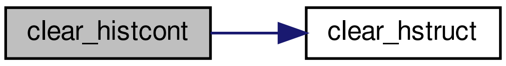
Returns

Number of elements released.

References history_container_struct::cfg_global, history_container_struct::cfg_tel, clear_hstruct(), history_↩ container_struct::cmdline, history_container_struct::id, and history_container_struct::ntel.

Referenced by read_history().

Here is the call graph for this function:



7.38.2.3 clear_hstruct()

```
int clear_hstruct (
            HSTRUCT * h )
```

Clear and free all elements along one linked list of history elements.

Parameters

h	Start of linked list of HSTRUCTs.

Returns

Number of elements released.

References history_struct::next, and history_struct::time.

Referenced by clear_histcont().

7.38.2.4 clear_metaparam()

```
int clear_metaparam (
            MetaParamList * lst )
```

Clear a list of meta parameters.

All pointers along the linked list must be dynamically allocated.

Parameters

lst	The starting point of a linked list. Must exist.

Returns

0 (OK), -1 or other non-zero (problem)

References meta_param_list::first, meta_param_list::ident, meta_param_item::name, meta_param_item::next, and meta_param_item::value.

Referenced by fill_metaparam().

7.38.2.5 fill_metaparam()

```
int fill_metaparam (
            MetaParamList * lst,
            const char ** names,
            const char ** values,
            size_t npar,
            long id )
```

Fill in meta parameters to the linked list after (re-)initialising it.

Parameters

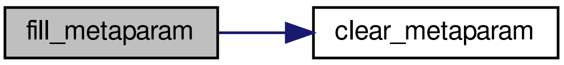
lst	The starting point of a linked list. Must exist but can be without items.
names	Array of names of the items to add. Array and all names must exist and be non-empty. @paran values Array of values of the items to add. Array and and values must exist (but can be empty).

Returns

0 (OK), -1 or other non-zero (problem)

References clear_metaparam(), meta_param_list::first, meta_param_list::ident, meta_param_item::name, meta_↩ param_item::next, and meta_param_item::value.

Here is the call graph for this function:



7.38.2.6 list_history()

```
int list_history (
            IO_BUFFER * iobuf,
            FILE * file )
```

List history block contents on standard output or other file.

Parameters

iobuf	I/O buffer descriptor
file	Optional open output stream (NULL -> stdout).

Returns

0 (o.k.), <0 (error with I/O buffer)

Referenced by print_history().

7.38.2.7 print_history()

```
int print_history (
              IO_BUFFER * iobuf )
```

Stub for list_history(), to stdout only.

Parameters

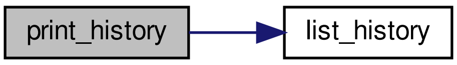
iobuf	I/O buffer descriptor

Returns

0 (o.k.), <0 (error with I/O buffer)

References list_history().

Here is the call graph for this function:



7.38.2.8 print_metaparam()

```
int print_metaparam (
              IO_BUFFER * iobuf )
```

Display the contents of a data block of meta parameters.

Parameters

iobuf	I/O buffer descriptor

Returns

0 (OK), -1 or other non-zero (problem)

7.38.2.9 push_command_history()

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

Save the command line for later output in a history block.

Parameters

argc	Number of command line arguments (incl. command)
argv	Pointers to argument text strings

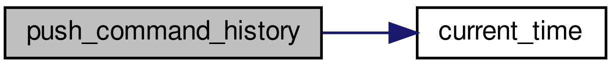
Returns

0 (o.k.), -1 (invalid argument or no memory)

References cmdline, cmdtime, and current_time().

Referenced by main().

Here is the call graph for this function:



7.38.2.10 push_config_history()

```
int push_config_history (
            const char * line,
            int noreplace )
```

Save a line of configuration text for later output in a history block.

If any configuration text for the same keyword was present before, it may be replaced by the new text, even if the new setting does not replace all old settings.

Parameters

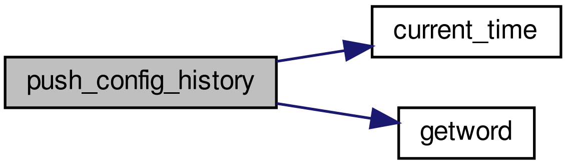
line	Configuration text line.
replace	Replace old text for same keyword (1) or not (0).

Returns

0 (o.k.), -1 (memory allocation failed)

References configs, current_time(), getword(), history_struct::next, and history_struct::time.

Here is the call graph for this function:



7.38.2.11 read_history()

```
int read_history (
            IO_BUFFER * iobuf,
            HistoryContainer * hc )
```

Read the block of accumulated history lines (command line and configuration lines) from an I/O buffer and try to split the configuration history by telescope, if the transition can be found.

Note that in extracted/split/merged sim_telarray data the correspondence to current telescope IDs is lost. For other programs, no telescope transition is recognized, i.e. all configuration is global.

Parameters

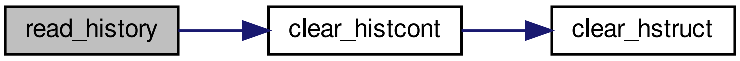
iobuf	I/O buffer descriptor
hc	Pointer to history container.

Returns

0 (o.k.), <0 (error with I/O buffer)

References clear_histcont().

Here is the call graph for this function:



7.38.2.12 read_metaparam()

```
int read_metaparam (
            IO_BUFFER * iobuf,
            MetaParamList * lst )
```

Read from a data block of meta parameters.

Parameters

iobuf	I/O buffer descriptor
lst	The starting point of a linked list. Must exist.

Returns

0 (OK), -1 or other non-zero (problem)

7.38.2.13 search_metaparam()

```
const char* search_metaparam (
            const MetaParamList * lst,
            const char * name )
```

Search for a specific metaparameter by name (case-insensitive, first match returned) in a linked list of meta parameters.

NULL is returned if there is no match.

Since metaparameters are stored only once per run (and telescope), and there should not be many of them, a linear search should be good enough.

Parameters

lst	The starting point of a linked list. Must exist but can be without items.
name	The name of the metaparameter searched for.

Returns

Value of metaparameter (first match found), or NULL pointer.

References meta_param_list::first, meta_param_item::name, meta_param_item::next, and meta_param_item↩ ::value.

7.38.2.14 show_metaparam()

```
int show_metaparam (
            const MetaParamList * lst )
```

Display the contents in a linked list of meta parameters, independent of its EventIO representation.

Parameters

lst	The starting point of a linked list. Must exist but can be without items.

Returns

0 (OK), -1 or other non-zero (problem)

References meta_param_list::first, meta_param_list::ident, meta_param_item::name, meta_param_item::next, and meta_param_item::value.

7.38.2.15 write_config_history()

```
int write_config_history (
            const char * htext,
            long htime,
            long id,
            IO_BUFFER * iobuf )
```

Write a configuration history line to an I/O buffer.

Parameters

htext	Text of configuration line
htime	Time when the configuration was set.
id	Identifier (detector number)
iobuf	I/O buffer descriptor

Returns

0 (o.k.), <0 (error with I/O buffer)

7.38.2.16 write_history()

```
int write_history (
            long id,
            IO_BUFFER * iobuf )
```

Write the block of accumulated history lines (command line and configuration lines) to an I/O buffer.

Parameters

id	Identifier (detector number)
iobuf	I/O buffer descriptor

Returns

0 (o.k.), <0 (error with I/O buffer)

7.38.2.17 write_metaparam()

```
int write_metaparam (
            IO_BUFFER * iobuf,
            const MetaParamList * lst )
```

Write a data block of meta parameters.

Parameters

iobuf	I/O buffer descriptor
lst	The starting point of a linked list. Must exist.

Returns

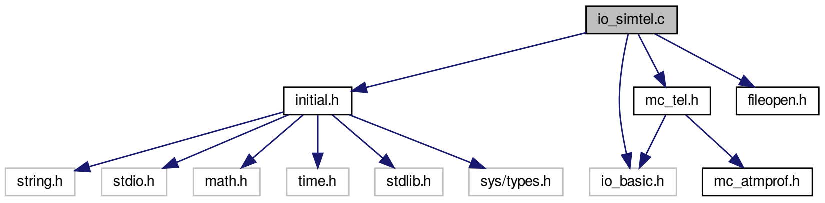
0 (OK), -1 or other non-zero (problem)

7.39 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

- static void **check_maxprt** (void)
- 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 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 print_tel_block (IO_BUFFER ∗iobuf)

  Print a CORSIKA header/trailer block of any type (see mc_tel.h)
- 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 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 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.
- 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 print_tel_pos (IO_BUFFER ∗iobuf)

  Print positions of telescopes/detectors within a system or array.
- 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 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 print_tel_offset (IO_BUFFER ∗iobuf)

  Print offsets and weights of randomly scattered arrays with respect to shower core.
- int begin_write_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int array)

  Begin writing 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 begin_read_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int ∗array)

  Begin reading data for one array of telescopes/detectors.
- int end_read_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih)

End reading 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_array_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int array)

  End writing 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_array_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int ∗array)

  End reading data for one array of telescopes/detectors.
- 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_photons3d (IO_BUFFER ∗iobuf, int array, int tel, double photons, struct bunch3d ∗bunches3d, int nbunches, int ext_bunches, char ∗ext_fname)

  Write all the photon bunches (3D) for one telescope to an I/O buffer.
- 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 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_photons3d (IO_BUFFER ∗iobuf, int max_bunches, int ∗array, int ∗tel, double ∗photons, struct bunch3d ∗bunches3d, int ∗nbunches)

  Read bunches of Cherenkov photons for one telescope/detector.
- int print_tel_photons (IO_BUFFER ∗iobuf)

  Print bunches of Cherenkov photons for one telescope/detector.
- int print_tel_photons3d (IO_BUFFER ∗iobuf)

  Print 3D bunches of Cherenkov photons for one telescope/detector.
- 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 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 print_shower_longitudinal (IO_BUFFER ∗iobuf)

  Print CORSIKA shower longitudinal distributions.
- 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 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 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 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 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 print_photo_electrons (IO_BUFFER ∗iobuf)

  List the the photoelectrons registered in a Cherenkov telescope camera.
- int **write_shower_extra_parameters** (IO_BUFFER ∗iobuf, ShowerExtraParam ∗ep)

- int **read_shower_extra_parameters** (IO_BUFFER ∗iobuf, ShowerExtraParam ∗ep)
- int **print_shower_extra_parameters** (IO_BUFFER ∗iobuf)
- int init_shower_extra_parameters (ShowerExtraParam ∗ep, size_t ni_max, size_t nf_max)

  Initialize, resize, clear shower extra parameters.
- int clear_shower_extra_parameters (ShowerExtraParam ∗ep)

  Similar to init_shower_extra_parameters() but without any attempts to re-allocate or resize buffers.
- ShowerExtraParam ∗ **get_shower_extra_parameters** ()
- 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 read_atmprof (IO_BUFFER ∗iobuf, AtmProf ∗atmprof)

  Read the atmospheric profile table as used in CORSIKA.
- int print_atmprof (IO_BUFFER ∗iobuf)

  Print the atmospheric profile table as used in CORSIKA.

Variables

- static int **max_print** = 10
- 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.

7.39.1 Detailed Description

Write and read CORSIKA blocks and simulated Cherenkov photon bunches.

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 2023

7.39.2 Function Documentation

7.39.2.1 begin_read_tel_array()

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

Begin reading data for one array of telescopes/detectors.

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

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

Returns

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

Referenced by array_conv_mc_phot(), array_select_mc_phot(), my_print_simtel_mc_phot(), and print_simtel_mc↩
_phot().

7.39.2.2 begin_write_tel_array()

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

Begin writing data for one array of telescopes/detectors.

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

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

Returns

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

7.39.2.3 clear_shower_extra_parameters()

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

Similar to init_shower_extra_parameters() but without any attempts to re-allocate or resize buffers.

Just clear contents.

Parameters

ep	Pointer to parameter block. A NULL value indicates that the static block is meant.

7.39.2.4 end_read_tel_array()

```
int end_read_tel_array (
            IO_BUFFER * iobuf,
            IO_ITEM_HEADER * ih )
```

End reading data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.39.2.5 end_write_tel_array()

```
int end_write_tel_array (
            IO_BUFFER * iobuf,
            IO_ITEM_HEADER * ih )
```

End writing data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.39.2.6 init_shower_extra_parameters()

```
int init_shower_extra_parameters (
            ShowerExtraParam * ep,
            size_t ni_max,
            size_t nf_max )
```

Initialize, resize, clear shower extra parameters.

Parameters

ep	Pointer to parameter block. A NULL value indicates that the static block is meant.
ni_max	The number of integer parameters to be used.
nf_max	The number of float parameters to be used.

7.39.2.7 print_atmprof()

```
int print_atmprof (
            IO_BUFFER * iobuf )
```

Print the atmospheric profile table as used in CORSIKA.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.8 print_camera_layout()

```
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.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.9 print_photo_electrons()

```
int print_photo_electrons (
            IO_BUFFER * iobuf )
```

List the the photoelectrons registered in a Cherenkov telescope camera.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.10 print_shower_longitudinal()

```
int print_shower_longitudinal (
            IO_BUFFER * iobuf )
```

Print CORSIKA shower longitudinal distributions.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.11 print_tel_block()

```
int print_tel_block (
            IO_BUFFER * iobuf )
```

Print a CORSIKA header/trailer block of any type (see mc_tel.h)

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.12 print_tel_offset()

```
int print_tel_offset (
            IO_BUFFER * iobuf )
```

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

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.13 print_tel_photons()

```
int print_tel_photons (
            IO_BUFFER * iobuf )
```

Print bunches of Cherenkov photons for one telescope/detector.

The data format may be either the more or less compact one.

Parameters

iobuf	I/O buffer descriptor

Returns

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

References compact_bunch::photons.

7.39.2.14 print_tel_photons3d()

```
int print_tel_photons3d (
            IO_BUFFER * iobuf )
```

Print 3D bunches of Cherenkov photons for one telescope/detector.

This is specific to the 3D format/data model.

Parameters

iobuf	I/O buffer descriptor

Returns

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

References bunch::photons.

7.39.2.15 print_tel_pos()

```
int print_tel_pos (
            IO_BUFFER * iobuf )
```

Print positions of telescopes/detectors within a system or array.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.39.2.16 read_atmprof()

```
int read_atmprof (
            IO_BUFFER * iobuf,
            AtmProf * atmprof )
```

Read the atmospheric profile table as used in CORSIKA.

Parameters

iobuf	I/O buffer descriptor
atmprof	Address of struct with relevant parts of atmospheric profile table

Returns

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

7.39.2.17 read_camera_layout()

```
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.

Parameters

iobuf	I/O buffer descriptor
max_pixels	The maximum number of pixels that can be stored in xp, yp.
itel	telescope number
type	camera type (hex/square)
pixels	number of pixels
xp	X positions of pixels
yp	Y position of pixels

Returns

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

7.39.2.18 read_input_lines()

```
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.

Parameters

iobuf	I/O buffer descriptor
list	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)

7.39.2.19 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 )
```

Read the photoelectrons registered in a Cherenkov telescope camera.

Parameters

iobuf	I/O buffer descriptor
max_pixels	Maximum number of pixels which can be treated
max_pe	Maximum number of photo-electrons
array	Array number
tel	Telescope number
npe	The total number of photo-electrons read.
pixels	Number of pixels read.
flags	Bit 0: amplitudes available, bit 1: includes NSB p.e.
pe_counts	Numbers of photo-electrons in each pixel
tstart	Offsets in 't' at which data for each pixel starts
t	Time of arrival of photons at the camera.
a	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
photon_counts	Optional number of photons arriving at a pixel.

Returns

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

7.39.2.20 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 )
```

Read CORSIKA shower longitudinal distributions.

See tellng_() in iact.c for more detailed parameter description.

Parameters

iobuf	I/O buffer descriptor
event	return event number
type	return 1 = particle numbers, 2 = energy, 3 = energy deposits
data	return set of (usually 9) distributions
ndim	maximum number of entries per distribution
np	return number of distributions (usually 9)
nthick	return number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
thickstep	return step size in g/cm**2
max_np	maximum number of distributions for which we have space.

Returns

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

7.39.2.21 read_tel_array_end()

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

End reading data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.39.2.22 read_tel_array_head()

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

Begin reading data for one array of telescopes/detectors.

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

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

Returns

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

7.39.2.23 read_tel_block()

```
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)

Parameters

iobuf	I/O buffer descriptor
type	block type (see mc_tel.h)
data	area for data to be read
maxlen	maximum number of elements to be read

Returns

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

7.39.2.24 read_tel_offset()

```
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.

Parameters

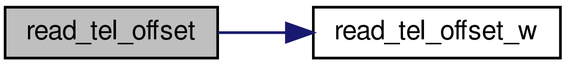
iobuf	I/O buffer descriptor
max_array	Maximum number of arrays that can be treated
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays

Returns

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

References read_tel_offset_w().

Here is the call graph for this function:



7.39.2.25 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 )
```

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

Parameters

iobuf	I/O buffer descriptor
max_array	Maximum number of arrays that can be treated

Parameters

narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays
weight	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)

Referenced by read_tel_offset().

7.39.2.26 read_tel_photons()

```
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.

The data format may be either the more or less compact one.

Parameters

iobuf	I/O buffer descriptor
max_bunches	maximum number of bunches that can be treated
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches	list of photon bunches
nbunches	number of elements in bunch list

Returns

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

Referenced by tel_conv_mc_phot().

7.39.2.27 read_tel_photons3d()

```
int read_tel_photons3d (
            IO_BUFFER * iobuf,
            int max_bunches,
            int * array,
            int * tel,
            double * photons,
            struct bunch3d * bunches3d,
            int * nbunches )
```

Read bunches of Cherenkov photons for one telescope/detector.

This is specific to the 3D format/data model.

Parameters

iobuf	I/O buffer descriptor
max_bunches	maximum number of bunches that can be treated
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches3d	list of 3D photon bunches
nbunches	number of elements in bunch list

Returns

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

7.39.2.28 read_tel_pos()

```
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.

Parameters

iobuf	I/O buffer descriptor
max_tel	maximum number of telescopes allowed
ntel	number of telescopes/detectors
x	X positions
y	Y positions
z	Z positions
r	radius of spheres including the whole devices

Returns

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

7.39.2.29 write_atmprof()

```
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.

Parameters

iobuf	I/O buffer descriptor
atmprof	Address of struct with relevant parts of atmospheric profile table

Returns

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

References atmospheric_profile::alt_km, atmospheric_profile::n_alt, atmospheric_profile::refidx_m1, atmospheric_profile::rho, and atmospheric_profile::thick.

7.39.2.30 write_camera_layout()

```
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.

Parameters

iobuf	I/O buffer descriptor
itel	telescope number
type	camera type (hex/square)
pixels	number of pixels
xp	X positions of pixels
yp	Y position of pixels

Returns

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

7.39.2.31 write_input_lines()

```
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.

Parameters

iobuf	I/O buffer descriptor
list	starting point of linked list

Returns

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

7.39.2.32 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 )
```

Write the photo-electrons registered in a Cherenkov telescope camera.

Parameters

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
npe	Total number of photo-electrons in the camera.
pixels	No. of pixels to be written
flags	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.

Parameters

pe_counts	Numbers of photo-electrons in each pixel
tstart	Offsets in 't' at which data for each pixel starts
t	Time of arrival of photons at the camera.
a	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
photon_counts	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)

7.39.2.33 write_shower_longitudinal()

```
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.

See tellng_() in iact.c for more detailed parameter description.

Parameters

iobuf	I/O buffer descriptor
event	event number
type	1 = particle numbers, 2 = energy, 3 = energy deposits
data	set of (usually 9) distributions
ndim	maximum number of entries per distribution
np	number of distributions (usually 9)
nthick	number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
thickstep	step size in g/cm∗∗2

Returns

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

7.39.2.34 write_tel_array_end()

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

End writing data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.39.2.35 write_tel_array_head()

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

Begin writing data for one array of telescopes/detectors.

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

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

Returns

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

7.39.2.36 write_tel_block()

```
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).

Parameters

iobuf	I/O buffer descriptor
type	block type (see mc_tel.h)
num	Run or event number depending on type
data	Data as passed from CORSIKA
len	Number of elements to be written

Returns

0 (OK), -1, -2, -3 (error, as usual in eventio)

7.39.2.37 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 )
```

Write all the photon bunches for one telescope to an I/O buffer.

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

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
cbunches	list of photon bunches
nbunches	number of elements in bunch list
ext_bunches	number of elements in external file
ext_fname	name of external (temporary) file

Returns

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

7.39.2.38 write_tel_offset()

```
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.

Parameters

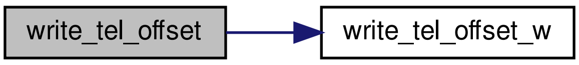
iobuf	I/O buffer descriptor
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays

Returns

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

References write_tel_offset_w().

Here is the call graph for this function:



7.39.2.39 write_tel_offset_w()

```
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.

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

iobuf	I/O buffer descriptor
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays
weight	Area weight for uniform or importance sampled core offset.

Returns

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

Referenced by write_tel_offset().

7.39.2.40 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 )
```

Write all the photon bunches for one telescope to an I/O buffer.

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

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches	list of photon bunches
nbunches	number of elements in bunch list
ext_bunches	number of elements in external file
ext_fname	name of external (temporary) file

Returns

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

7.39.2.41 write_tel_photons3d()

```
int write_tel_photons3d (
            IO_BUFFER * iobuf,
            int array,
            int tel,
            double photons,
            struct bunch3d * bunches3d,
            int nbunches,
            int ext_bunches,
            char * ext_fname )
```

Write all the photon bunches (3D) for one telescope to an I/O buffer.

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 complete 3D format (40 bytes per bunch). Note that while the normal bunch and compact bunch variants use a different data format but the same data model (and thus can come as variants of the same data block type), the 3D variant has a different format as well as a different data model and therefore needs a different data block type.

Parameters

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches3d	list of 3D photon bunches
nbunches	number of elements in bunch list
ext_bunches	number of elements in external file
ext_fname	name of external (temporary) file

Returns

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

7.39.2.42 write_tel_pos()

```
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.

Parameters

iobuf	I/O buffer descriptor

Parameters

ntel	number of telescopes/detectors
x	X positions
y	Y positions
z	Z positions
r	radius of spheres including the whole devices

Returns

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

7.39.3 Variable Documentation

7.39.3.1 private_shower_extra_parameters

```
ShowerExtraParam private_shower_extra_parameters  [static]
```

There is one global (more precisely: static) block of extra shower parameters as, for example, used in the CORSIKA IACT interface.

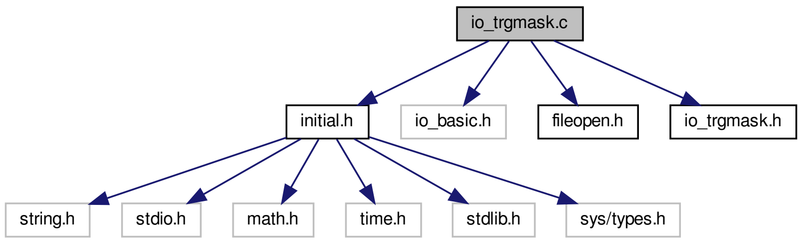
Get a pointer to this block.

7.40 io_trgmask.c File Reference

EventIO plus helper functions for trigger type bit patterns extracted from sim_telarray log files (only relevant for simulations with multiple trigger types using sim_telarray versions before mid-2013).

```
#include "initial.h"
#include "io_basic.h"
#include "fileopen.h"
#include "io_trgmask.h"
```

Include dependency graph for io_trgmask.c:



Macros

- #define **TMS_ALLOCS** 100

Functions

- int trgmask_scan_log (struct trgmask_set ∗tms, const char ∗fname)

  Scan a sim_telarray log file for lines related to trigger type mask bit patterns.
- int write_trgmask (IO_BUFFER ∗iobuf, struct trgmask_set ∗tms)

  Write the accumulated trigger mask bit patterns as an I/O block.
- int print_trgmask (IO_BUFFER ∗iobuf)

  Print the trigger mask bit patterns contained in an I/O block.
- int read_trgmask (IO_BUFFER ∗iobuf, struct trgmask_set ∗tms)

  Read the trigger mask bit patterns contained in an I/O block.
- int trgmask_fill_hashed (struct trgmask_set ∗tms, struct trgmask_hash_set ∗ths)

  Fill an array of linked lists of trgmask entries, suitable for hashing.
- struct trgmask_entry ∗ find_trgmask (struct trgmask_hash_set ∗ths, long event, int tel_id)

  Find the trgmask entry for a given event and telescope in the hashed list.
- void print_hashed_trgmasks (struct trgmask_hash_set ∗ths)

  Print the collected trgmask entries in the order as hashed.

7.40.1 Detailed Description

EventIO plus helper functions for trigger type bit patterns extracted from sim_telarray log files (only relevant for simulations with multiple trigger types using sim_telarray versions before mid-2013).

Author

Konrad Bernloehr

Date

2013 to 2022

7.40.2 Function Documentation

7.40.2.1 find_trgmask()

```
struct trgmask_entry* find_trgmask (
            struct trgmask_hash_set * ths,
            long event,
            int tel_id )
```

Find the trgmask entry for a given event and telescope in the hashed list.

Hash collisions are handled by linear search through the linked list at each hash entry.

Parameters

ths	The trgmask hash set.
event	The event number in the search.
tel↩_id	The telescope ID in the search.

Returns

A pointer to the trgmask entry searched for, or NULL for not found.

7.40.2.2 print_hashed_trgmasks()

```
void print_hashed_trgmasks (
            struct trgmask_hash_set * ths )
```

Print the collected trgmask entries in the order as hashed.

Also show the maximum number of colliding entries under one hash value.

7.40.2.3 trgmask_fill_hashed()

```
int trgmask_fill_hashed (
            struct trgmask_set * tms,
            struct trgmask_hash_set * ths )
```

Fill an array of linked lists of trgmask entries, suitable for hashing.

Hash collisions are handled by linear search through the linked list at each hash entry.

7.40.2.4 trgmask_scan_log()

```
int trgmask_scan_log (
            struct trgmask_set * tms,
            const char * fname )
```

Scan a sim_telarray log file for lines related to trigger type mask bit patterns.

Parameters

tms	The trigger mask structure into which results should be filled in.
fname	The name of the log file to be opened.

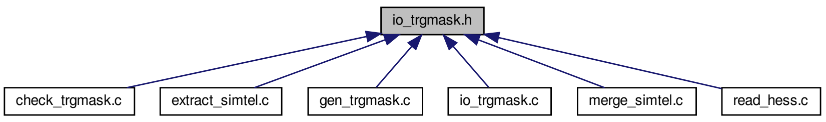
Returns

0 (OK), -1 (invalid parameters or file not found), -2 (allocation error, partially filled)

7.41 io_trgmask.h File Reference

EventIO plus helper functions for trigger type bit patterns extracted from sim_telarray log files (only relevant for simulations with multiple trigger types using sim_telarray versions before mid-2013).

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



Data Structures

- struct trgmask_entry
- struct trgmask_set
- struct trgmask_hash_set

Macros

- #define IO_TYPE_HESS_XTRGMASK 2090

  Extra (or external - not in normal data file) trigger mask data block type.
- #define **IO_TYPE_SIMTEL_XTRGMASK** 2090
- #define **TRGMASK_PRIME** 15269
- #define **TRGMASK_HASH**(ev, ti) (((ti)∗10000+(ev))%TRGMASK_PRIME)

Functions

- int trgmask_scan_log (struct trgmask_set ∗tms, const char ∗fname)

  Scan a sim_telarray log file for lines related to trigger type mask bit patterns.
- int write_trgmask (IO_BUFFER ∗iobuf, struct trgmask_set ∗tms)

  Write the accumulated trigger mask bit patterns as an I/O block.
- int print_trgmask (IO_BUFFER ∗iobuf)

  Print the trigger mask bit patterns contained in an I/O block.
- int read_trgmask (IO_BUFFER ∗iobuf, struct trgmask_set ∗tms)

  Read the trigger mask bit patterns contained in an I/O block.
- int trgmask_fill_hashed (struct trgmask_set ∗tms, struct trgmask_hash_set ∗ths)

  Fill an array of linked lists of trgmask entries, suitable for hashing.
- struct trgmask_entry ∗ find_trgmask (struct trgmask_hash_set ∗ths, long event, int tel_id)

  Find the trgmask entry for a given event and telescope in the hashed list.
- void print_hashed_trgmasks (struct trgmask_hash_set ∗ths)

  Print the collected trgmask entries in the order as hashed.

7.41.1 Detailed Description

EventIO plus helper functions for trigger type bit patterns extracted from sim_telarray log files (only relevant for simulations with multiple trigger types using sim_telarray versions before mid-2013).

Date

2013, 2022

7.41.2 Function Documentation

7.41.2.1 find_trgmask()

```
struct trgmask_entry* find_trgmask (
             struct trgmask_hash_set * ths,
             long event,
             int tel_id )
```

Find the trgmask entry for a given event and telescope in the hashed list.

Hash collisions are handled by linear search through the linked list at each hash entry.

Parameters

ths	The trgmask hash set.
event	The event number in the search.
tel←_id	The telescope ID in the search.

Returns

A pointer to the trgmask entry searched for, or NULL for not found.

7.41.2.2 print_hashed_trgmasks()

```
void print_hashed_trgmasks (
             struct trgmask_hash_set * ths )
```

Print the collected trgmask entries in the order as hashed.

Also show the maximum number of colliding entries under one hash value.

7.41.2.3 trgmask_fill_hashed()

```
int trgmask_fill_hashed (
            struct trgmask_set * tms,
            struct trgmask_hash_set * ths )
```

Fill an array of linked lists of trgmask entries, suitable for hashing.

Hash collisions are handled by linear search through the linked list at each hash entry.

7.41.2.4 trgmask_scan_log()

```
int trgmask_scan_log (
            struct trgmask_set * tms,
            const char * fname )
```

Scan a sim_telarray log file for lines related to trigger type mask bit patterns.

Parameters

tms	The trigger mask structure into which results should be filled in.
fname	The name of the log file to be opened.

Returns

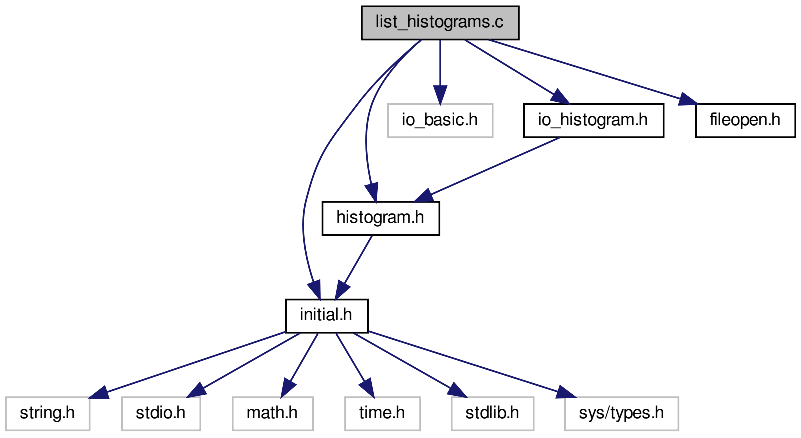
0 (OK), -1 (invalid parameters or file not found), -2 (allocation error, partially filled)

7.42 list_histograms.c File Reference

Utility program for listing histograms and extracting histogram data.

```
#include "initial.h"
#include "histogram.h"
#include "io_basic.h"
#include "io_histogram.h"
#include "fileopen.h"
```

Include dependency graph for list_histograms.c:



Functions

- long project_histogram (long ihisto, int proj)

  Project a 2-D histogram onto one of its axes.
- void print_ratio (HISTOGRAM ∗histo1, HISTOGRAM ∗histo2, double fact)

  Print ratio of two histograms: fact ∗ histo1 / histo2.
- int main (int argc, char ∗∗argv)

  Main program.

7.42.1 Detailed Description

Utility program for listing histograms and extracting histogram data.

```
Syntax:  list_histograms [ input_file ... ]
```

The default input file name is 'testpattern.hdata'. The histograms may be within multiple I/O blocks of the input file.

Author

Konrad Bernloehr

Date

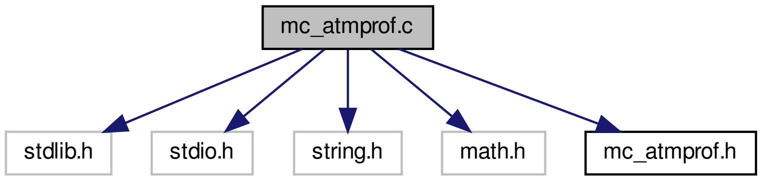
2001 to 2023

7.43 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:



Functions

- AtmProf ∗ get_common_atmprof (void)

  Make this copy of the atmospheric profile available elsewhere.
- 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.
- 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)
- double rhofc (double ∗height)

  C-called functions equivalent to the CORSIKA-built-in functions to evaluate the 5-layer parametrization.
- double **thickc** (double ∗height)
- double **refidc** (double ∗height)
- double **refim1c** (double ∗height)
- double **heighc** (double ∗thick)

Variables

- static AtmProf common_atmprof

  Keep track of atmospheric profiles loaded from text tables.
- static double **etadsn0** = 0.000283 ∗ 994186.38 / 1222.656

7.43.1 Detailed Description

Interface to the atmospheric profile structure.

Author

Konrad Bernloehr

Date

2019, 2020

7.43.2 Function Documentation

7.43.2.1 rhofc()

```
double rhofc (
            double * height )
```

C-called functions equivalent to the CORSIKA-built-in functions to evaluate the 5-layer parametrization.

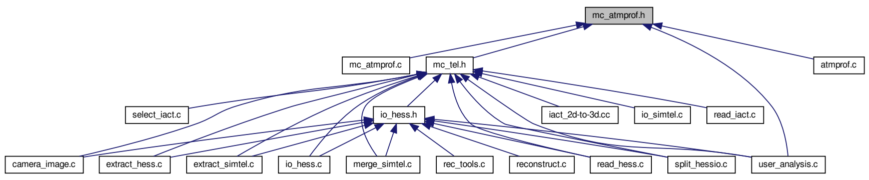
Assumes that these parameters have been set before. Where the numerical table is available it should be used once to initialize the atmospheric profile and then use the corresponding rhofx_(), ... functions for the evaulation instead.

References atmospheric_profile::batm, common_atmprof, atmospheric_profile::datm, and atmospheric_profile↩
::hlay.

7.44 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

- AtmProf ∗ get_common_atmprof (void)

  Make this copy of the atmospheric profile available elsewhere.
- 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.
- 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)
- double rhofc (double ∗height)

  C-called functions equivalent to the CORSIKA-built-in functions to evaluate the 5-layer parametrization.
- double **thickc** (double ∗height)
- double **refidc** (double ∗height)
- double **refim1c** (double ∗height)
- double **heighc** (double ∗thick)

7.44.1 Detailed Description

A data structure shared between io_simtel.c and atmo.c - which is used by both sim_telarray and the CORSIKA IACT/atmo package.

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

7.44.2 Function Documentation

7.44.2.1 rhofc()

```
double rhofc (
            double * height )
```

C-called functions equivalent to the CORSIKA-built-in functions to evaluate the 5-layer parametrization.

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 evaulation instead.

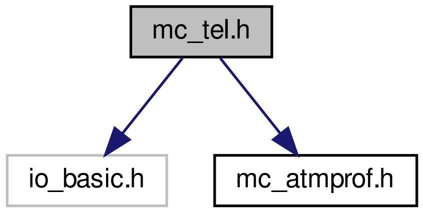
References atmospheric_profile::batm, common_atmprof, atmospheric_profile::datm, and atmospheric_profile↵ ::hlay.

7.45 mc_tel.h File Reference

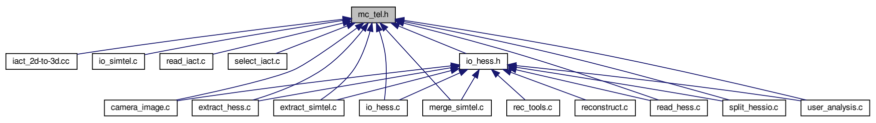
Definitions and structures for CORSIKA Cherenkov light interface.

```
#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 bunch3d

  A more complete, alternative bunch structure which can also represent upward-going photon bunches or horizontal ones while the bunch and cbunch structures strictly assume downward-going photon bunches.
- struct compact_bunch

  The compact_bunch struct is equivalent to the bunch struct except that we try to use less memory.
- struct photo_electron

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

  The linked_string is mainly used to keep CORSIKA input.
- struct shower_extra_parameters

  Extra shower parameters of unspecified nature.

Macros

- #define **_MC_TEL_LOADED** 2
- #define **IO_TYPE_MC_BASE** 1200
- #define **IO_TYPE_MC_RUNH** (IO_TYPE_MC_BASE+0)
- #define **IO_TYPE_MC_TELPOS** (IO_TYPE_MC_BASE+1)
- #define **IO_TYPE_MC_EVTH** (IO_TYPE_MC_BASE+2)
- #define **IO_TYPE_MC_TELOFF** (IO_TYPE_MC_BASE+3)
- #define **IO_TYPE_MC_TELARRAY** (IO_TYPE_MC_BASE+4)
- #define **IO_TYPE_MC_PHOTONS** (IO_TYPE_MC_BASE+5)
- #define **IO_TYPE_MC_LAYOUT** (IO_TYPE_MC_BASE+6)
- #define **IO_TYPE_MC_TRIGTIME** (IO_TYPE_MC_BASE+7)
- #define **IO_TYPE_MC_PE** (IO_TYPE_MC_BASE+8)
- #define **IO_TYPE_MC_EVTE** (IO_TYPE_MC_BASE+9)
- #define **IO_TYPE_MC_RUNE** (IO_TYPE_MC_BASE+10)
- #define **IO_TYPE_MC_LONGI** (IO_TYPE_MC_BASE+11)
- #define **IO_TYPE_MC_INPUTCFG** (IO_TYPE_MC_BASE+12)
- #define **IO_TYPE_MC_TELARRAY_HEAD** (IO_TYPE_MC_BASE+13)
- #define **IO_TYPE_MC_TELARRAY_END** (IO_TYPE_MC_BASE+14)
- #define **IO_TYPE_MC_EXTRA_PARAM** (IO_TYPE_MC_BASE+15)
- #define **IO_TYPE_MC_ATMPROF** (IO_TYPE_MC_BASE+16)
- #define **IO_TYPE_MC_PHOTONS3D** (IO_TYPE_MC_BASE+17)

Typedefs

- typedef float **real**
- typedef short **INT16**
- typedef unsigned short **UINT16**
- typedef int **INT32**
- typedef unsigned int **UINT32**
- typedef struct shower_extra_parameters **ShowerExtraParam**

Functions

- 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 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 print_tel_block (IO_BUFFER ∗iobuf)

  Print a CORSIKA header/trailer block of any type (see mc_tel.h)
- 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 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 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.
- 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 print_tel_pos (IO_BUFFER ∗iobuf)

  Print positions of telescopes/detectors within a system or array.
- 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 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 print_tel_offset (IO_BUFFER ∗iobuf)

  Print offsets and weights of randomly scattered arrays with respect to shower core.
- int begin_write_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int array)

  Begin writing 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 begin_read_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int ∗array)

  Begin reading data for one array of telescopes/detectors.
- int end_read_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih)

  End reading 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_array_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int array)

  End writing 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_array_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int ∗array)

  End reading data for one array of telescopes/detectors.
- 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_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 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 print_tel_photons (IO_BUFFER ∗iobuf)

    Print bunches of Cherenkov photons for one telescope/detector.
- int write_tel_photons3d (IO_BUFFER ∗iobuf, int array, int tel, double photons, struct bunch3d ∗bunches3d, int nbunches, int ext_bunches, char ∗ext_fname)

    Write all the photon bunches (3D) for one telescope to an I/O buffer.
- int read_tel_photons3d (IO_BUFFER ∗iobuf, int max_bunches, int ∗array, int ∗tel, double ∗photons, struct bunch3d ∗bunches3d, int ∗nbunches)

    Read bunches of Cherenkov photons for one telescope/detector.
- int print_tel_photons3d (IO_BUFFER ∗iobuf)

    Print 3D bunches of Cherenkov photons for one telescope/detector.
- 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 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 print_shower_longitudinal (IO_BUFFER ∗iobuf)

    Print CORSIKA shower longitudinal distributions.
- 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 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 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 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 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 print_photo_electrons (IO_BUFFER ∗iobuf)

    List the the photoelectrons registered in a Cherenkov telescope camera.
- int **write_shower_extra_parameters** (IO_BUFFER ∗iobuf, ShowerExtraParam ∗ep)
- int **read_shower_extra_parameters** (IO_BUFFER ∗iobuf, ShowerExtraParam ∗ep)
- int **print_shower_extra_parameters** (IO_BUFFER ∗iobuf)
- int init_shower_extra_parameters (ShowerExtraParam ∗ep, size_t ni_max, size_t nf_max)

    Initialize, resize, clear shower extra parameters.
- int clear_shower_extra_parameters (ShowerExtraParam ∗ep)

    Similar to init_shower_extra_parameters() but without any attempts to re-allocate or resize buffers.
- ShowerExtraParam ∗ **get_shower_extra_parameters** (void)
- void **remember_corsika_atm_params** (float ∗hlay, float ∗aatm, float ∗batm, float ∗catm)
- int **get_corsika_atm_params** (double ∗hlay, double ∗aatm, double ∗batm, double ∗catm, double ∗datm)
- 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 read_atmprof (IO_BUFFER ∗iobuf, AtmProf ∗atmprof)

    Read the atmospheric profile table as used in CORSIKA.
- int print_atmprof (IO_BUFFER ∗iobuf)

    Print the atmospheric profile table as used in CORSIKA.

7.45.1 Detailed Description

Definitions and structures for CORSIKA Cherenkov light interface.

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 2023

7.45.2 Function Documentation

7.45.2.1 begin_read_tel_array()

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

Begin reading data for one array of telescopes/detectors.

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

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

Returns

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

Referenced by array_conv_mc_phot(), array_select_mc_phot(), my_print_simtel_mc_phot(), and print_simtel_mc↵_phot().

7.45.2.2 begin_write_tel_array()

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

Begin writing data for one array of telescopes/detectors.

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

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

Returns

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

7.45.2.3 clear_shower_extra_parameters()

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

Similar to init_shower_extra_parameters() but without any attempts to re-allocate or resize buffers.

Just clear contents.

Parameters

ep	Pointer to parameter block. A NULL value indicates that the static block is meant.

7.45.2.4 end_read_tel_array()

```
int end_read_tel_array (
            IO_BUFFER * iobuf,
            IO_ITEM_HEADER * ih )
```

End reading data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.45.2.5 end_write_tel_array()

```
int end_write_tel_array (
            IO_BUFFER * iobuf,
            IO_ITEM_HEADER * ih )
```

End writing data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.45.2.6 init_shower_extra_parameters()

```
int init_shower_extra_parameters (
            ShowerExtraParam * ep,
            size_t ni_max,
            size_t nf_max )
```

Initialize, resize, clear shower extra parameters.

Parameters

ep	Pointer to parameter block. A NULL value indicates that the static block is meant.
ni_max	The number of integer parameters to be used.
nf_max	The number of float parameters to be used.

7.45.2.7 print_atmprof()

```
int print_atmprof (
            IO_BUFFER * iobuf )
```

Print the atmospheric profile table as used in CORSIKA.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.8 print_camera_layout()

```
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.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.9 print_photo_electrons()

```
int print_photo_electrons (
            IO_BUFFER * iobuf )
```

List the the photoelectrons registered in a Cherenkov telescope camera.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.10 print_shower_longitudinal()

```
int print_shower_longitudinal (
            IO_BUFFER * iobuf )
```

Print CORSIKA shower longitudinal distributions.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.11 print_tel_block()

```
int print_tel_block (
             IO_BUFFER * iobuf )
```

Print a CORSIKA header/trailer block of any type (see mc_tel.h)

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.12 print_tel_offset()

```
int print_tel_offset (
             IO_BUFFER * iobuf )
```

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

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.13 print_tel_photons()

```
int print_tel_photons (
             IO_BUFFER * iobuf )
```

Print bunches of Cherenkov photons for one telescope/detector.

The data format may be either the more or less compact one.

Parameters

iobuf	I/O buffer descriptor

Returns

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

References compact_bunch::photons.

7.45.2.14 print_tel_photons3d()

```
int print_tel_photons3d (
            IO_BUFFER * iobuf )
```

Print 3D bunches of Cherenkov photons for one telescope/detector.

This is specific to the 3D format/data model.

Parameters

iobuf	I/O buffer descriptor

Returns

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

References bunch::photons.

7.45.2.15 print_tel_pos()

```
int print_tel_pos (
            IO_BUFFER * iobuf )
```

Print positions of telescopes/detectors within a system or array.

Parameters

iobuf	I/O buffer descriptor

Returns

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

7.45.2.16 read_atmprof()

```
int read_atmprof (
            IO_BUFFER * iobuf,
            AtmProf * atmprof )
```

Read the atmospheric profile table as used in CORSIKA.

Parameters

iobuf	I/O buffer descriptor
atmprof	Address of struct with relevant parts of atmospheric profile table

Returns

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

7.45.2.17 read_camera_layout()

```
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.

Parameters

iobuf	I/O buffer descriptor
max_pixels	The maximum number of pixels that can be stored in xp, yp.
itel	telescope number
type	camera type (hex/square)
pixels	number of pixels
xp	X positions of pixels
yp	Y position of pixels

Returns

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

7.45.2.18 read_input_lines()

```
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.

Parameters

iobuf	I/O buffer descriptor
list	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)

7.45.2.19 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 )
```

Read the photoelectrons registered in a Cherenkov telescope camera.

Parameters

iobuf	I/O buffer descriptor
max_pixels	Maximum number of pixels which can be treated
max_pe	Maximum number of photo-electrons
array	Array number

Parameters

tel	Telescope number
npe	The total number of photo-electrons read.
pixels	Number of pixels read.
flags	Bit 0: amplitudes available, bit 1: includes NSB p.e.
pe_counts	Numbers of photo-electrons in each pixel
tstart	Offsets in 't' at which data for each pixel starts
t	Time of arrival of photons at the camera.
a	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
photon_counts	Optional number of photons arriving at a pixel.

Returns

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

7.45.2.20 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 )
```

Read CORSIKA shower longitudinal distributions.

See tellng_() in iact.c for more detailed parameter description.

Parameters

iobuf	I/O buffer descriptor
event	return event number
type	return 1 = particle numbers, 2 = energy, 3 = energy deposits
data	return set of (usually 9) distributions
ndim	maximum number of entries per distribution
np	return number of distributions (usually 9)
nthick	return number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
thickstep	return step size in g/cm∗∗2
max_np	maximum number of distributions for which we have space.

Returns

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

7.45.2.21 read_tel_array_end()

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

End reading data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.45.2.22 read_tel_array_head()

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

Begin reading data for one array of telescopes/detectors.

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

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

Returns

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

7.45.2.23 read_tel_block()

```
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)

Parameters

iobuf	I/O buffer descriptor
type	block type (see mc_tel.h)
data	area for data to be read
maxlen	maximum number of elements to be read

Returns

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

7.45.2.24 read_tel_offset()

```
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.

Parameters

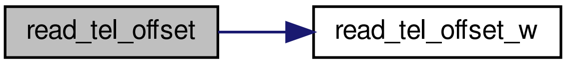
iobuf	I/O buffer descriptor
max_array	Maximum number of arrays that can be treated
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays

Returns

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

References read_tel_offset_w().

Here is the call graph for this function:



7.45.2.25 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 )
```

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

Parameters

iobuf	I/O buffer descriptor
max_array	Maximum number of arrays that can be treated
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays
weight	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)

Referenced by read_tel_offset().

7.45.2.26 read_tel_photons()

```
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.

The data format may be either the more or less compact one.

Parameters

iobuf	I/O buffer descriptor
max_bunches	maximum number of bunches that can be treated
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches	list of photon bunches
nbunches	number of elements in bunch list

Returns

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

Referenced by tel_conv_mc_phot().

7.45.2.27 read_tel_photons3d()

```
int read_tel_photons3d (
            IO_BUFFER * iobuf,
            int max_bunches,
            int * array,
            int * tel,
            double * photons,
            struct bunch3d * bunches3d,
            int * nbunches )
```

Read bunches of Cherenkov photons for one telescope/detector.

This is specific to the 3D format/data model.

Parameters

iobuf	I/O buffer descriptor
max_bunches	maximum number of bunches that can be treated
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches3d	list of 3D photon bunches
nbunches	number of elements in bunch list

Returns

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

7.45.2.28 read_tel_pos()

```
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.

Parameters

iobuf	I/O buffer descriptor
max_tel	maximum number of telescopes allowed
ntel	number of telescopes/detectors
x	X positions
y	Y positions
z	Z positions
r	radius of spheres including the whole devices

Returns

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

7.45.2.29 write_atmprof()

```
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.

Parameters

iobuf	I/O buffer descriptor
atmprof	Address of struct with relevant parts of atmospheric profile table

Returns

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

References atmospheric_profile::alt_km, atmospheric_profile::n_alt, atmospheric_profile::refidx_m1, atmospheric←_profile::rho, and atmospheric_profile::thick.

7.45.2.30 write_camera_layout()

```
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.

Parameters

iobuf	I/O buffer descriptor
itel	telescope number
type	camera type (hex/square)
pixels	number of pixels
xp	X positions of pixels
yp	Y position of pixels

Returns

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

7.45.2.31 write_input_lines()

```
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.

Parameters

iobuf	I/O buffer descriptor
list	starting point of linked list

Returns

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

7.45.2.32 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 )
```

Write the photo-electrons registered in a Cherenkov telescope camera.

Parameters

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
npe	Total number of photo-electrons in the camera.
pixels	No. of pixels to be written
flags	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.
pe_counts	Numbers of photo-electrons in each pixel
tstart	Offsets in 't' at which data for each pixel starts
t	Time of arrival of photons at the camera.
a	Amplitudes of p.e. signals [mean p.e.] (optional, may be NULL).
photon_counts	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)

7.45.2.33 write_shower_longitudinal()

```
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.

See tellng_() in iact.c for more detailed parameter description.

Parameters

iobuf	I/O buffer descriptor
event	event number
type	1 = particle numbers, 2 = energy, 3 = energy deposits
data	set of (usually 9) distributions
ndim	maximum number of entries per distribution
np	number of distributions (usually 9)
nthick	number of entries actually filled per distribution (is 1 if called without LONGI being enabled).
thickstep	step size in g/cm∗∗2

Returns

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

7.45.2.34 write_tel_array_end()

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

End writing data for one array of telescopes/detectors.

Parameters

iobuf	I/O buffer descriptor
ih	I/O item header (as opened in begin_write_tel_array())

Returns

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

7.45.2.35 write_tel_array_head()

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

Begin writing data for one array of telescopes/detectors.

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

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

Returns

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

7.45.2.36 write_tel_block()

```
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).

Parameters

iobuf	I/O buffer descriptor
type	block type (see mc_tel.h)
num	Run or event number depending on type
data	Data as passed from CORSIKA
len	Number of elements to be written

Returns

0 (OK), -1, -2, -3 (error, as usual in eventio)

7.45.2.37 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 )
```

Write all the photon bunches for one telescope to an I/O buffer.

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

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
cbunches	list of photon bunches
nbunches	number of elements in bunch list
ext_bunches	number of elements in external file
ext_fname	name of external (temporary) file

Returns

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

7.45.2.38 write_tel_offset()

```
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.

Parameters

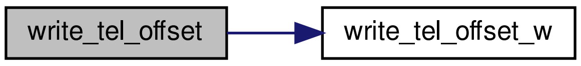
iobuf	I/O buffer descriptor
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays

Returns

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

References write_tel_offset_w().

Here is the call graph for this function:



7.45.2.39 write_tel_offset_w()

```
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.

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

iobuf	I/O buffer descriptor
narray	Number of arrays of telescopes/detectors
toff	Time offset (ns, from first interaction to ground)
xoff	X offsets of arrays
yoff	Y offsets of arrays
weight	Area weight for uniform or importance sampled core offset.

Returns

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

Referenced by write_tel_offset().

7.45.2.40 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 )
```

Write all the photon bunches for one telescope to an I/O buffer.

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

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches	list of photon bunches
nbunches	number of elements in bunch list
ext_bunches	number of elements in external file
ext_fname	name of external (temporary) file

Returns

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

7.45.2.41 write_tel_photons3d()

```
int write_tel_photons3d (
             IO_BUFFER * iobuf,
             int array,
             int tel,
             double photons,
             struct bunch3d * bunches3d,
             int nbunches,
             int ext_bunches,
             char * ext_fname )
```

Write all the photon bunches (3D) for one telescope to an I/O buffer.

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 complete 3D format (40 bytes per bunch). Note that while the normal bunch and compact bunch variants use a different data format but the same data model (and thus can come as variants of the same data block type), the 3D variant has a different format as well as a different data model and therefore needs a different data block type.

Parameters

iobuf	I/O buffer descriptor
array	array number
tel	telescope number
photons	sum of photons (and fractions) in this device
bunches3d	list of 3D photon bunches
nbunches	number of elements in bunch list
ext_bunches	number of elements in external file
ext_fname	name of external (temporary) file

Returns

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

7.45.2.42 write_tel_pos()

```
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.

Parameters

iobuf	I/O buffer descriptor
ntel	number of telescopes/detectors
x	X positions
y	Y positions
z	Z positions
r	radius of spheres including the whole devices

Returns

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

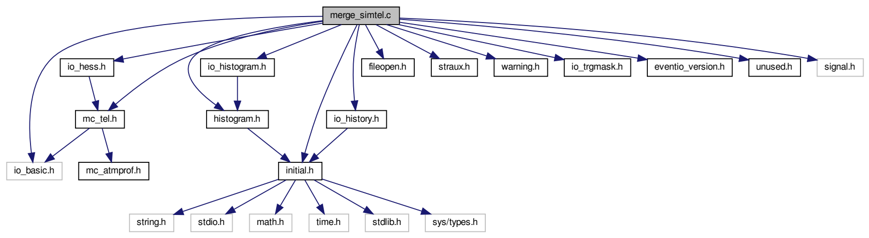
7.46 merge_simtel.c File Reference

A program for merging events from separate telescope simulations of the same showers.

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

```
#include "mc_tel.h"
#include "io_history.h"
#include "io_hess.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
#include "straux.h"
#include "warning.h"
#include "io_trgmask.h"
#include "eventio_version.h"
#include "unused.h"
#include <signal.h>
```

Include dependency graph for merge_simtel.c:



Data Structures

- struct map_tel_struct

  Structure with per output telescope information keeping track of prerequisites.

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- int find_in_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the input structures.
- int find_out_tel_idx (int tel_id, int ifile)

  Offset of an input telescope of given ID within the output structures.
- int find_mapped_telescope (int tel_id, int ifile)

  Mapping from telescope ID on input to telescope ID on output, with check.
- int write_io_block_to_file (IO_BUFFER ∗iobuf, FILE ∗f)

  Write an I/O block as-is to another file than foreseen for the I/O buffer.
- int **has_min_trg_tel** (AllHessData ∗hsdata_out, int mtrg, double rtm)
- int check_for_delayed_write (IO_ITEM_HEADER ∗item_header, _unused_ int ifile, AllHessData ∗hsdata_out, IO_BUFFER ∗iobuf_out)

  Check if previously delayed writing of output should be done now.
- int merge_data_from_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header, int ifile, AllHessData ∗hsdata, AllHessData ∗hsdata_out, IO_BUFFER ∗iobuf_out)

  Processing and merging of I/O blocks from the two input files, hopefully presented in the right order.
- int check_autoload_trgmask (const char ∗input_fname, IO_BUFFER ∗iobuf, int ifile)

  Check for a 'trgmask.gz' file matching the given input data file name and, if it exists, extract the corrected trigger bit patterns from it.

- void **print_process_status** (int prev_type1, int this_type1, int prev_type2, int this_type2)
- int **read_map** (const char ∗map_fname)
- static void syntax (const char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**
- static int **verbose** = 0
- struct map_tel_struct **map_tel** [H_MAX_TEL]
- int map_to [2][H_MAX_TEL+1]

  Mapping structures from input telescope ID to output telescope ID.
- int tel_idx [2][H_MAX_TEL+1]

  Mapping from telescope IDs to offsets in the data structures, first for input telescope IDs.
- int tel_idx_out [H_MAX_TEL+1]

  Mapping from output telescope ID to offset in output data structures.
- int **ntel1**
- int **ntel2**
- int **ntel**
- int **nrtel1**
- int **nrtel2**
- long **event1** = -1
- long **event2** = 0
- long **ev_hess_event** = 0
- long ev_pe_sum = 0

  For delayed writing.
- int **run1** = -1
- int **run2** = -1
- int **min_trg** = 2
- double **distinct_sep** = 1.0
- static struct trgmask_set ∗ **tms** [2] = { NULL, NULL }
- static struct trgmask_hash_set ∗ **ths** [2] = { NULL, NULL }
- static int **events** [2] = { 0, 0 }
- static int **mcshowers** [2] = { 0, 0 }
- static int **mcevents** [2] = { 0, 0 }
- static int **max_list** = 999

7.46.1 Detailed Description

A program for merging events from separate telescope simulations of the same showers.

The program will read sim_telarray raw or DST data on two input files, map telescope ID according to a mapping file and write the merged blocks to an output file.

Inputs expected - and the action to be performed: Type Once per run: 70 (history) - Write as-is, impossible to merge 2000 (run_header) - Merging needed for telescope list and positions 2001 (MC run header) - Only one of two MC run-headers needed (should be identical) 1212 (input config = CORSIKA inputs) - Only one needed (should be identical, duplicate) 1216 (atmospheric density profile) - Only one needed (should be identical, duplicate) Once per telescope (and per run for raw & DST levels 0-2; just once for DST level 3): 75 (metaparam) - Write after mapping of telescope ID (if mapped); global remains ID -1. 2002 (camera settings) - Write after mapping of telescope ID (if mapped) 2003 (camera organization) - Write after mapping of telescope ID (if mapped) 2004 (pixel settings) - Write after mapping of telescope ID (if mapped) 2005 (pixel disable) - Write after mapping of telescope ID (if mapped) 2006 (camera software settings) - Write after mapping of telescope ID (if mapped) 2008 (tracking settings) - Write after mapping of telescope ID (if mapped) 2007 (pointing corrections) - Write after mapping of telescope ID (if mapped) 2022 (telescope monitoring) - Write after mapping of telescope ID (if mapped) 2023 (Laser calibration) - Write after mapping of telescope ID (if mapped) 2033 (MC pixel monitoring) - Write after mapping of telescope ID (if mapped) Per shower: once: 2020 (MC shower) - Only one of two MC run-headers needed (should be identical) per array: 2021 (MC event) - Only one of two blocks needed (anything to get merged?) Optional per event; not immediately written but delayed until next MC etc. block: 2026 (MC pe sum) - ??? 1204 (photo-electrons individually) - ??? 2010 (event) - Needs remapping and merging at all levels At end of run: 2024 (run statistics - usually not present) 2025 (MC run statistics - usually not present) 100 (histograms) - Cannot be merged properly. Histograms of generated showers should agree, but for triggered showers we cannot tell how many are common.

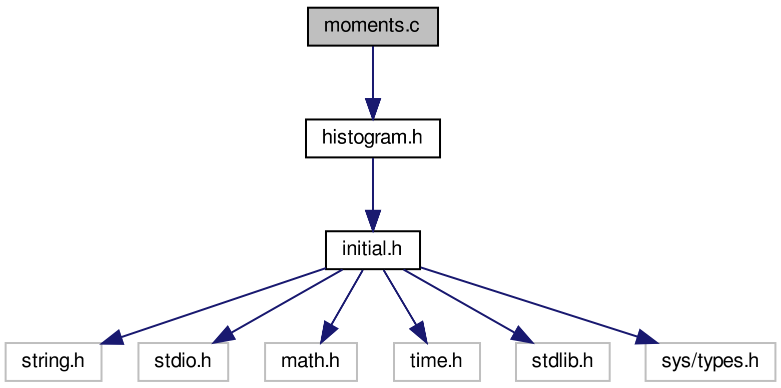
FIXME: Ignoring 'trgmask' files initially - include them later on.

```
Syntax:  merge_simtel [ options ] map-file input1 input2 output
Options:
     --auto-trgmask  : Load trgmask.gz files for each input file where available.
     --min-trg-tel n : Require at least n telescopes in merged event (default: 2).
     --verbose       : Show events being merged.


@author  Konrad Bernloehr
@date    2013 to 2023
```

7.47 moments.c File Reference

Calculate mean, rms, skewness, and kurtosis of data.

`#include "histogram.h"`
Include dependency graph for moments.c:



Functions

- MOMENTS ∗ alloc_moments (HISTVALUE_REAL low, HISTVALUE_REAL high)

    Allocate a structure for sums of powers of data.
- void clear_moments (MOMENTS ∗mom)

    Initialize an existing moments structure (except for its range limits).
- void free_moments (MOMENTS ∗mom)

    Deallocates memory previously allocated to a moments structure.
- void fill_moments (MOMENTS ∗mom, HISTVALUE_REAL value)

    Add up those things needed to compute mean, standard deviation, skewness, and kurtosis (both for all data and separately for data in a range defined in alloc_moments().
- void fill_mean_and_sigma (MOMENTS ∗mom, HISTVALUE_REAL value)

    Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().
- void fill_mean (MOMENTS ∗mom, HISTVALUE_REAL value)

    Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().
- void fill_real_moments (MOMENTS ∗mom, HISTVALUE_REAL value, double weight)

    Add up those things needed to compute – mean, – standard deviation, – skewness, and – kurtosis (both for all data and separately for data in a range defined in alloc_moments().
- void fill_real_mean_and_sigma (MOMENTS ∗mom, HISTVALUE_REAL value, double weight)

    Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().
- void fill_real_mean (MOMENTS ∗mom, HISTVALUE_REAL value, double weight)

    Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().
- int stat_moments (MOMENTS ∗mom, struct momstat ∗stmom)

    Calculate moments (mean, rms, skewness, kurtosis) from the sums of powers of data values.

7.47.1 Detailed Description

Calculate mean, rms, skewness, and kurtosis of data.

Author

Konrad Bernloehr

Date

1995 to 2011

7.47.2 Function Documentation

7.47.2.1 alloc_moments()

```
MOMENTS* alloc_moments (
            HISTVALUE_REAL low,
            HISTVALUE_REAL high )
```

Allocate a structure for sums of powers of data.

Returns NULL if no structure could be allocated.

Parameters

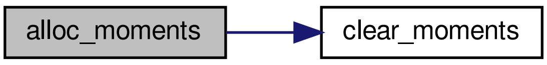
low	Lower limit of range for truncation
high	Upper limit of range for truncation

Returns

Pointer to allocated structure or NULL.

References clear_moments().

Here is the call graph for this function:



7.47.2.2 clear_moments()

```
void clear_moments (
            MOMENTS * mom )
```

Initialize an existing moments structure (except for its range limits).

Parameters

mom	Pointer to moments structure

Referenced by alloc_moments().

7.47.2.3 fill_mean()

```
void fill_mean (
            MOMENTS * mom,
            HISTVALUE_REAL value )
```

Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value

7.47.2.4 fill_mean_and_sigma()

```
void fill_mean_and_sigma (
            MOMENTS * mom,
            HISTVALUE_REAL value )
```

Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value

7.47.2.5 fill_moments()

```
void fill_moments (
            MOMENTS * mom,
            HISTVALUE_REAL value )
```

Add up those things needed to compute mean, standard deviation, skewness, and kurtosis (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value

7.47.2.6 fill_real_mean()

```
void fill_real_mean (
            MOMENTS * mom,
            HISTVALUE_REAL value,
            double weight )
```

Add up those things needed to compute – mean, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value
weight	Weighting factor of this value

7.47.2.7 fill_real_mean_and_sigma()

```
void fill_real_mean_and_sigma (
            MOMENTS * mom,
            HISTVALUE_REAL value,
            double weight )
```

Add up those things needed to compute – mean, – standard deviation, (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value
weight	Weighting factor of this value

7.47.2.8 fill_real_moments()

```
void fill_real_moments (
            MOMENTS * mom,
            HISTVALUE_REAL value,
            double weight )
```

Add up those things needed to compute – mean, – standard deviation, – skewness, and – kurtosis (both for all data and separately for data in a range defined in alloc_moments().

Parameters

mom	Pointer to previously allocated MOMENTS structure.
value	One measurement value
weight	Weighting factor of this value

7.47.2.9 free_moments()

```
void free_moments (
            MOMENTS * mom )
```

Deallocates memory previously allocated to a moments structure.

Parameters

mom	Pointer to previously allocated structure

7.47.2.10 stat_moments()

```
int stat_moments (
            MOMENTS * mom,
            struct momstat * stmom )
```

Calculate moments (mean, rms, skewness, kurtosis) from the sums of powers of data values.

Parameters

mom	'moments' structure with the sums of the powers of data values (only 1st power if only mean to be calculated, also 2nd power if r.m.s. to be calculated, and also 3rd and 4th if skewness and kurtosis wanted.
stmom	Pointer to structure for computed moments

Returns

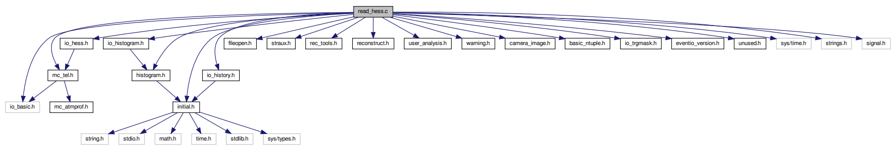
0 (o.k.), -1 and -2 (invalid data)

7.48 read_hess.c File Reference

A program reading simulated data, optionally analysing the data, and also optionally also writing summary ("DST") data.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_history.h"
#include "io_hess.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
#include "straux.h"
#include "rec_tools.h"
#include "reconstruct.h"
#include "user_analysis.h"
#include "warning.h"
#include "camera_image.h"
#include "basic_ntuple.h"
#include "io_trgmask.h"
#include "eventio_version.h"
#include "unused.h"
#include <sys/time.h>
#include <strings.h>
```

```
#include <signal.h>
```
Include dependency graph for read_hess.c:



Data Structures

- struct next_file_struct
- struct range_list_struct

Macros

- #define CALIB_SCALE 0.92

  The factor needed to transform from mean p.e.
- #define **_XSTR_**(s) _STR_(s)
- #define **_STR_**(s) #s
- #define **SHOW**(s) if (strcmp(#s,_XSTR_(s)) != 0) printf(" " #s " = " _XSTR_(s) "\n")

Typedefs

- typedef struct next_file_struct **NextFile**
- typedef struct range_list_struct **RangeList**

Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void **init_rand** (int is)
- double grand48 (double mean, double sigma)

  Like RandFlat() from rndm2.c but using the drand48 engine.
- static void mc_event_fill (AllHessData ∗hsdata, double d_sp_idx)

  Fill histogram(s) for DST writing which require all MC shower and event data and which cannot be filled from DST level >= 2 data.
- static int write_dst_histos (IO_BUFFER ∗iobuf2)

  Write histograms for DST book-keeping and clear them afterwards.
- static void **show_run_summary** (AllHessData ∗hsdata, int nev, int ntrg, double plidx, double wsum_all, double wsum_trg, double rmax_x, double rmax_y, double rmax_r)
- static void syntax (char ∗program)

  Show program syntax.
- NextFile ∗ **add_next_file** (const char ∗fn, NextFile ∗nxt)
- RangeList ∗ **add_range** (long f, long t, RangeList ∗rl)
- int **is_in_range** (long n, RangeList ∗rl)
- int **read_disabled_pixels_list** (const char ∗fname, PixelDisabled ∗∗list)
- void show_header (IO_ITEM_HEADER ∗item_header)

  Print (to stdout) what information we have in the item header.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- struct basic_ntuple **bnt**
- static int **interrupted**
- static int **dst_processing**
- static int **g48_set**
- static double **g48_next**

7.48.1 Detailed Description

A program reading simulated data, optionally analysing the data, and also optionally also writing summary ("DST") data.

This program started as a skeleton for reading H.E.S.S. data in eventio format (which is what the read_simtel_nr program is now intended for). The read_hess program reads the whole range of hessio item types into a single tree of data structures but normally does nothing with the data.

It can be instructed to create nice camera images similar to those generated in sim_hessarray.

It can also be instructed to redo the image cleaning (with the simple 10/5 tail-cut algorithm) and the shower reconstruction, writing ASCII output of the results.

In addition, it includes an interface for a full-scale analysis which can optionally be activated.

And finally, it can be instructed to extract DST-level data in order to reduce the amount of data by a large factor. This depends on the dst-level flag: 1) Remove all raw data (you cannot redo image cleaning) afterwards. 2) Remove also all MC data from non-triggered event (you should better stay with the spectral index used for DST extraction because you have to rely on its histograms for MC energy distribution). 3) and 4) Keep only user-defined events (with or without raw data).

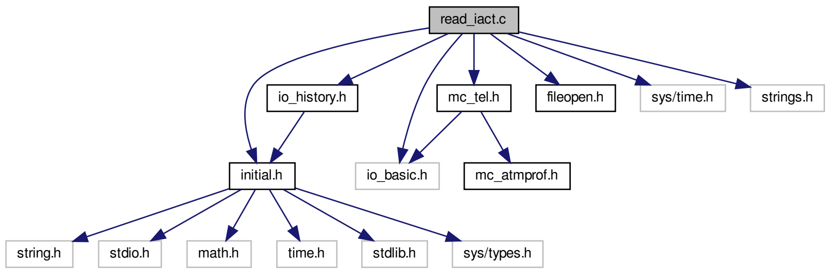
```
read_hess: A program for viewing and analyzing sim_telarray (sim_hessarray) data.

Syntax: read_hess [ options ] [ - | input_fname ... ]
Options:
   -p ps_filename  (Write a PostScript file with camera images.)
   --plot-with-true-pe (If data available, include true p.e. plot in PS file.)
   --plot-with-sum-only (Show only sum image even if we have traces.)
   --plot-with-pixel-id (Show pixel ID number on top of pixel.)
   --plot-with-pixel-amp (Show pixel amplitude value on top of pixel.)
   --plot-with-pixel-pe (Show count of true Cherenkov p.e. on top of pixel.)
   --plot-without-reco (Do not show reconstructed image/shower parameters.)
   --plot-with-type-sum (Plot sum of pixel intensities over telescope types.)
   --plot-with-title text (User-defined title on top of page.)
   -r level        (Use 10/5 tail-cut image cleaning and redo reconstruction.)
                    level >= 1: show parameters from sim_hessarray.
                    level >= 2: redo shower reconstruction
                    level >= 3: redo image cleaning (and shower reconstruction
                                with new image parameters)
                    level >= 4: redo amplitude summation
                    level >= 5: PostScript file includes original and
                                new shower reconstruction.
   -v              (More verbose output)
   -q              (Much more quiet output)
   -s              (Show data explained)
   -S              (Show data explained, including raw data)
   --history (-h)  (Show contents of history data block)
   --clean-history (Drop previous history data blocks)
   -i              (Ignore unknown data block types)
   -u              (Call user-defined analysis function)
   --global-peak   (For image analysis use amplitude sums around global peak
                     in 'on-line' pulse shape analysis.)
   --local-peak    (For image analysis use amplitude sums around local peaks
                 in 'on-line' pulse shape analysis.)
   --powerlaw x    (Use this spectral index for events weights in output.)
                   (Default spectral index is -2.7)
   --only-event ev1[,ev2-ev3[,...]] (Select specific events.)
   --only-shower n1[,n2-n3[,...]] (Select specific showers.)
   --only-run run1[,run2-run3[,...]] (Select runs being processed.)
   --not-run run1[,run2-run3[,...]]
   --only-type t1[,t2[,...]] (Select telescopes of given type(s) only.)
   --only-telescope id1[,id2-i3[,...]] (Select telescopes being used by ID.)
   --not-telescope id1[,id2-id3[,...]]
   --required-telescope id (A specific telescope which has to have data.)
   --auto-trgmask  (Automatically load matching .trgmask.gz files.)
   --trgmask-path dir (Search the trgmask files in this path first.)
   --trg-required b *(Required trigger bits, e.g. 5=1|4 -> majo or asum)
   --type nt[,id1,id2,A,f,npix] (Set [requirements for] telescope type nt.)
   --focal-length f *(Set telescope imaging effective focal length [m].)
   --min-tel tmn   *(The minimum number of tel. images required in analysis.)
   --max-tel tmx   (The maximum number of tel. images required in analysis.)
   --min-trg-tel n (Minimum number of telescopes in system trigger.)
   --max-trg-tel n (Maximum number of telescopes in system trigger.)
   --hard-stereo id1,id2,.. (Telescope of ID id1 etc. only use if stereo.)
   --min-amp npe   *(Minimum image amplitude for shower reconstruction.)
   --min-pix npix  *(Minimum number of pixels for shower reconstruction.)
   --max-events n  (Skip remaining data after so many triggered events.)
   --max-theta d   (Maximum angle between source and shower direction [deg].)
   --min-theta d   (Where cut angle is multiplicity dependent, use this
                    as the lower limit [deg].)
   --theta-scale f (Scale fixed and optimized theta cut by this factor.)
   --theta-E-scale t0,ts,min,max (Energy-dependent scaling beyond multiplicity.)
   --tail-cuts l,h[,n,f] *(Low and high level tail cuts to be applied in analysis.)
   --nb-radius r1[,r2[,r3]] *(Maximum distance of neighbour pixels [px diam.])
   --ext-radius r  *(Radius to extend preserved pixels beyond cleaning [px diam.])
   --dE2-cut c     (Cut parameter for dE2 cut.)
   --hess-standard-cuts (Apply HESS-style selection with standard cuts.)
   --hess-hard-cuts (Apply HESS-style selection with hard cuts.)
   --hess-loose-cuts (Apply HESS-style selection with loose cuts.)
   --hess-style-cuts (No shape parameter rescaling as HESS-style.)
   --shape-cuts wmn,wmx,lmn,lmx (Shape cut parameters: mscrw/l min/max).
   --dE-cut c      (Scale parameter for dE cut strictness, def=1.0).
   --hmax-cut c    (Scale parameter for hmax cut strictness, def=1.0).
   --min-img-angle a (Only use image pairs intersecting at angle > a deg, def=0).
   --min-disp d    *(Do not use round images with disp = (1-w/l) < d, def=0).
   --max-core-distance r *(Only use images from telescope not further from core).
   --impact-range r,x,y (Accept only events with reconstructed core in range).
   --true-impact-range r,x,y (Accept only events with true core in range).
                    Note that r is in shower plane but x,y ranges are on surface.
   --min-true-energy e (Completely skip events below given true energy.
   --clip-camera-radius r *(In image reconstruction clip camera at radius r deg.)
   --clip-camera-diameter d *(Same as before but with diameter d deg.)
   --clip-pixel-amplitude a *(Calibrated pixel ampl. does not exceed a mean p.e.)
   --only-high-gain (Use only high-gain channel and ignore low gain.)
   --only-low-gain (Use only low-gain channel and ignore high gain.)
   --max-events    (Stop after having processed this many events.)
   --pure-raw      (Discard any sub-items of TelescopeEvent which are not raw data.)
   --no-mc-data    (Discard MC shower and MC event data.)
   --broken-pixels-fraction (Add random broken/dead pixels on run-by-run basis.)
   --broken-pixels-list (Replace broken pixels with pre-generated lists.)
   --dead-time-fraction (Set telescopes randomly as dead from prior triggers.)
   --integration-scheme n *(Set the integration scheme for sample-mode data.
                   Use '--integration-scheme help' to show available schemes.)
   --integration-window w,o[,ps] *(Set integration window width and offset.)
                   For some integration schemes there is a pulse shaping option.
   --integration-treshold h[,l] *(Set significance thresholds for integration.)
   --integration-no-rescale *(Don't rescale pulse sum for integration with
                   windows narrower than a single-p.e. pulse.)
   --integration-rescale *(Rescale for single-p.e. fraction in window; default)
   --calib-scale f *(Rescale from mean p.e. to experiment units. Default: 0.92)
   --calib-error f (Random pixel relative calibration error. Default: 0.)
   --calibrate     (Store calibrated pixel intensities to DST file, if possible.)
   --only-calibrated (Like '--calibrate' but omit raw data from DST.)
   --pixel-stats   *(Fill histograms of pixel trigger statistics.)
   --diffuse-mode  (True shower position assumed as source position.)
   --random-seed n|auto (Initialize random number generator.)
   --off-axis-range a1,a2 (Only for diffuse mode, restricting range in deg.)
   --auto-lookup   (Automatically generate lookup table (gammas only).)
   --lookup-file name (Override automatic naming of lookup files.)
   --cleaning n    (Imaging cleaning setting: 0=no, 1-5=yes, see '--cleaning help')
   --zero-suppression n (Zero suppression scheme; 0: off, 3=auto)
   -z              (Equivalent to '--zero-suppression auto')
   --dst-level n   (Level of data reduction when writing DST-type output.)
                   Valid levels: 0, 1, 2, 3, 10, 11, 12, 13.
                   Raw data is stripped off at all levels except 0 and 10.
                   Level 0 has any sample mode data reduced to sums,
                   Level 1 includes all MC shower/event blocks,
                   level 2 only for triggered events,
                   level 3 has many config/calib blocks only once, not per run.
                   Levels 10-13 include only selected gamma-like events.
   --raw-level n   (Re-write original raw data or processed data, with possible
                   selection or reduction of other data according to level.)
                   Level 0 has all data written as available.
                   Level 1 has MC data only for triggered events.
                   Level 2 has no MC data (--no-mc-data).
                   Level 3 has only raw data for telescopes and nothing else (--pure-raw).
                   Level 4 also cleans past history data (--clean-history).
   --dst-file name (Name of output file for DST-type output.)
                   A DST file is needed for cleaning > 0 or DST level >= 0.
   --output-file   (Synonym to --dst-file)
   --histogram-file name (Name of histogram file.)
   -f fname        (Get list of input file names from fname.)

Parameters followed by a '*' can be type-specific if preceded by a
'--type' option. Their interpretation is thus position-dependent.


@author Konrad Bernloehr

@date    2001 to 2023
```

7.49 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 "io_history.h"
#include "mc_tel.h"
#include "fileopen.h"
#include <sys/time.h>
#include <strings.h>
```

Include dependency graph for read_iact.c:



Functions

- int my_print_simtel_mc_phot (IO_BUFFER ∗iobuf)

  Print Monte Carlo photons and photo-electrons.
- void **syntax** (void)
- void **show_header** (IO_ITEM_HEADER ∗item_header)
- int main (int argc, char ∗∗argv)

  Main program.

7.49.1 Detailed Description

A program reading simulated CORSIKA data written through the IACT interface and shows the contents as readable text.

Relevant environment variables: PRINT_TEL_VERBOSE MAX_PRINT_ARRAY

Author

Konrad Bernloehr

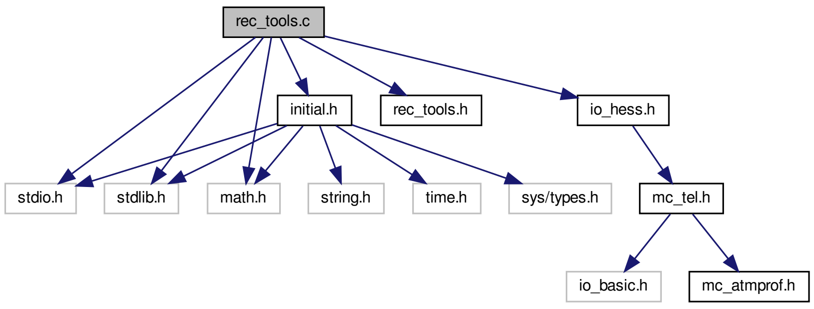
Date

2018 to 2023

7.50 rec_tools.c File Reference

Tools for shower geometric reconstruction.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "initial.h"
#include "rec_tools.h"
#include "io_hess.h"
```
Include dependency graph for rec_tools.c:



Macros

- #define **MAX_TEL** H_MAX_TEL
- #define **WT_DISP** 1

Functions

- double line_point_distance (double xp1, double yp1, double zp1, double cx, double cy, double cz, double x, double y, double z)

  Distance between a straight line and a point in space.
- void angles_to_offset (double obj_azimuth, double obj_altitude, double azimuth, double altitude, double focal_length, double ∗xoff, double ∗yoff)

  Transform telescope and object Alt/Az to offset in camera.
- void offset_to_angles (double xoff, double yoff, double azimuth, double altitude, double focal_length, double ∗obj_azimuth, double ∗obj_altitude)

  Transform from offset in camera to corresponding Az/Alt.
- void get_shower_trans_matrix (double azimuth, double altitude, double trans[ ][3])

  Calculate transformation matrix.
- void cam_to_ref (double ximg, double yimg, double phi, double ref_azimuth, double ref_altitude, double cam_rot, double azimuth, double altitude, double focal_length, double ∗axref, double ∗ayref, double ∗phiref)

  Transform from one camera to common reference frame.
- int intersect_lines (double xp1, double yp1, double phi1, double xp2, double yp2, double phi2, double ∗xs, double ∗ys, double ∗sang)

  Intersect pairs of lines.
- static double **square** (double a)
- int shower_geometric_reconstruction (int ntel, const double ∗amp, const double ∗ximg, const double ∗yimg, const double ∗phi, const double ∗disp, const double ∗xtel, const double ∗ytel, const double ∗ztel, const double ∗az, const double ∗alt, const double ∗flen, const double ∗cam_rot, double ref_az, double ref_alt, int flag, double ∗shower_az, double ∗shower_alt, double ∗var_dir, double ∗xc, double ∗yc, double ∗var_core)

  Simple reconstruction by intersecting pairs of lines.
- double angle_between (double azimuth1, double altitude1, double azimuth2, double altitude2)

  Calculate the angle between two directions given in spherical coordinates.

7.50.1 Detailed Description

Tools for shower geometric reconstruction.

Shower geometric reconstruction based on the major axes of the telescope images. The image parameters from each telescope are transformed to a common reference frame first before the average intersection point of all images is calculated in plane coordinates.

Author

Konrad Bernloehr

Date

2000 to 2019

7.50.2 Function Documentation

7.50.2.1 angle_between()

```
double angle_between (
            double azimuth1,
            double altitude1,
            double azimuth2,
            double altitude2 )
```

Calculate the angle between two directions given in spherical coordinates.

Returns

The angle between the two directions in units of radians.

7.50.2.2 angles_to_offset()

```
void angles_to_offset (
            double obj_azimuth,
            double obj_altitude,
            double azimuth,
            double altitude,
            double focal_length,
            double * xoff,
            double * yoff )
```

Transform telescope and object Alt/Az to offset in camera.

Transform from given telescope and object angles (Az/Alt) to the offset the object has in the camera plane.

This does not account for any rotation of the camera and its pixels.

Referenced by cam_to_ref().

7.50.2.3 cam_to_ref()

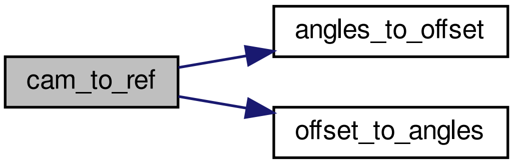
```
void cam_to_ref (
            double ximg,
            double yimg,
            double phi,
            double ref_azimuth,
            double ref_altitude,
            double cam_rot,
            double azimuth,
            double altitude,
            double focal_length,
            double * axref,
            double * ayref,
            double * phiref )
```

Transform from one camera to common reference frame.

Transform from the camera plane coordinate system of a telescope looking to altitude/azimuth to a plane coordinate system of a potential telescope looking to a reference direction ref_azimuth,ref_altitude and having unit focal length. Rotation of image angles is accounted for but not imaging errors.

References angles_to_offset(), and offset_to_angles().

Here is the call graph for this function:



7.50.2.4 get_shower_trans_matrix()

```
void get_shower_trans_matrix (
             double azimuth,
             double altitude,
             double trans[ ][3] )
```

Calculate transformation matrix.

Calculate transformation matrix from horizontal reference frame to one z axis in the given Az/Alt direction and the x axis in the plane defined by Az/Alt and zenith.

7.50.2.5 intersect_lines()

```
int intersect_lines (
             double xp1,
             double yp1,
             double phi1,
             double xp2,
             double yp2,
             double phi2,
             double * xs,
             double * ys,
             double * sang )
```

Intersect pairs of lines.

Intersect a pair of straight lines in a plane and return the intersection point and the angle at which the lines intersect.

7.50.2.6 line_point_distance()

```
double line_point_distance (
             double xp1,
             double yp1,
             double zp1,
             double cx,
             double cy,
             double cz,
             double x,
             double y,
             double z )
```

Distance between a straight line and a point in space.

Parameters

xp1,yp1,zp1	reference point on the line
cx,cy,cz	direction cosines of the line
x,y,z	point in space

Returns

distance

Referenced by mc_event_fill().

7.50.2.7 offset_to_angles()

```
void offset_to_angles (
             double xoff,
             double yoff,
             double azimuth,
             double altitude,
             double focal_length,
             double * obj_azimuth,
             double * obj_altitude )
```

Transform from offset in camera to corresponding Az/Alt.

Transform from the offset an object or image has in the camera plane of a telescope to the corresponding Az/Alt.

This does not account for any rotation of the camera and its pixels. (xoff and yoff are assumed to be corrected for camera rotation). In the presence of imaging errors, an effective focal length should be used.

Referenced by cam_to_ref().

7.50.2.8 shower_geometric_reconstruction()

```
int shower_geometric_reconstruction (
            int ntel,
            const double * amp,
            const double * ximg,
            const double * yimg,
            const double * phi,
            const double * disp,
            const double * xtel,
            const double * ytel,
            const double * ztel,
            const double * az,
            const double * alt,
            const double * flen,
            const double * cam_rot,
            double ref_az,
            double ref_alt,
            int flag,
            double * shower_az,
            double * shower_alt,
            double * var_dir,
            double * xc,
            double * yc,
            double * var_core )
```

Simple reconstruction by intersecting pairs of lines.

Simple geometric shower reconstruction by intersecting pairs of straigh lines (from major axis of second moments ellipses after transformation to a common plane), first for the shower direction and then for the core position. No errors on reconstructed direction or core position are calculated. This should sooner or later be superceded by a fit procedure taking advantage of estimated errors on image positions and angles.

Parameters

ntel	The number of telescopes with suitable images.
amp	The image amplitudes in each suitable telescope [p.e.].
ximg	The image c.o.g. x positions in the local camera coordinate systems.
yimg	The image c.o.g. y positions in the local camera coordinate systems.
phi	The image major axis direction [rad].
disp	The DISP parameter (1.-width/length), used for giving preference to elongated images. Set all to 1.0 if unknown or no preference wanted. Can also be passed as a NULL pointer instead.
xtel	The x coordinate of the telescope positions within array [m].
ytel	The y coordinate of the telescope positions within array [m].
ztel	The z coordinate of the telescope positions within array [m].
az	The azimuth angles to which the telescopes are pointing (N->E->S->W) [rad].
alt	The altitude angles to which the telescopes are pointing [rad].
flen	The focal length to which ximg and yimg are scaled (1.0 if in units of radians, otherwise flen is in meters).
cam_rot	Camera rotation angle [rad].
ref_az	The reference azimuth angle (system nominal azimuth) [rad].
ref_alt	The reference altitude angle (system nominal altitude) [rad].
flag	Use the reconstucted direction to derive the core position (0) or use the nominal direction for that (1 or any other non-zero). The second version may sightly improve core distance and thus energy accuracy for well-defined point sources.

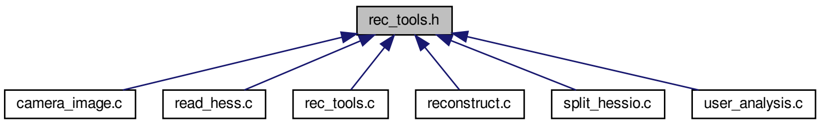
Parameters

shower_az	Return the reconstructed shower azimuth angle (N->E->S->W) [rad].
shower_alt	Return the reconstructed shower altitude angle [rad].
var_dir	Variance (dx∗∗2+dy∗∗2)/ntel of reconstructed direction for more than two images. Can be NULL if you are not interested in it.
xc	Return the reconstructed core position x coordinate (at z=0) [m].
yc	Return the reconstructed core position y coordinate (at z=0) [m].
var_core	Variance (dx∗∗2+dy∗∗2)/ntel of reconstructed core position for more than two images. Can be NULL if you are not interested in it.

7.51 rec_tools.h File Reference

Function prototypes for rec_tools.c.

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



Functions

- void angles_to_offset (double obj_azimuth, double obj_altitude, double azimuth, double altitude, double focal_length, double ∗xoff, double ∗yoff)

  Transform telescope and object Alt/Az to offset in camera.
- void offset_to_angles (double xoff, double yoff, double azimuth, double altitude, double focal_length, double ∗obj_azimuth, double ∗obj_altitude)

  Transform from offset in camera to corresponding Az/Alt.
- void get_shower_trans_matrix (double azimuth, double altitude, double trans[ ][3])

  Calculate transformation matrix.
- void cam_to_ref (double ximg, double yimg, double phi, double ref_azimuth, double ref_altitude, double cam_rot, double azimuth, double altitude, double focal_length, double ∗axref, double ∗ayref, double ∗phiref)

  Transform from one camera to common reference frame.
- int intersect_lines (double xp1, double yp1, double phi1, double xp2, double yp2, double phi2, double ∗xs, double ∗ys, double ∗sang)

  Intersect pairs of lines.
- int shower_geometric_reconstruction (int ntel, const double ∗amp, const double ∗ximg, const double ∗yimg, const double ∗phi, const double ∗disp, const double ∗xtel, const double ∗ytel, const double ∗ztel, const double ∗az, const double ∗alt, const double ∗flen, const double ∗cam_rot, double ref_az, double ref_alt, int flag, double ∗shower_az, double ∗shower_alt, double ∗var_dir, double ∗xc, double ∗yc, double ∗var_core)

  Simple reconstruction by intersecting pairs of lines.
- double angle_between (double azimuth1, double altitude1, double azimuth2, double altitude2)

  Calculate the angle between two directions given in spherical coordinates.
- double line_point_distance (double xp1, double yp1, double zp1, double cx, double cy, double cz, double x, double y, double z)

  Distance between a straight line and a point in space.

7.51.1 Detailed Description

Function prototypes for rec_tools.c.

Author

Konrad Bernloehr

Date

2001 to 2010

7.51.2 Function Documentation

7.51.2.1 angle_between()

```
double angle_between (
             double azimuth1,
             double altitude1,
             double azimuth2,
             double altitude2 )
```

Calculate the angle between two directions given in spherical coordinates.

Returns

The angle between the two directions in units of radians.

7.51.2.2 angles_to_offset()

```
void angles_to_offset (
             double obj_azimuth,
             double obj_altitude,
             double azimuth,
             double altitude,
             double focal_length,
             double * xoff,
             double * yoff )
```

Transform telescope and object Alt/Az to offset in camera.

Transform from given telescope and object angles (Az/Alt) to the offset the object has in the camera plane.

Transform from given telescope and object angles (Az/Alt) to the offset the object has in the camera plane.

This does not account for any rotation of the camera and its pixels.

Referenced by cam_to_ref().

7.51.2.3 cam_to_ref()

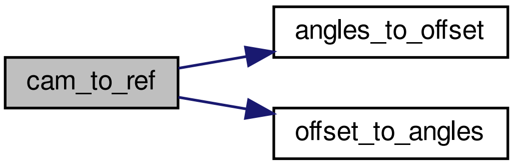
```
void cam_to_ref (
            double ximg,
            double yimg,
            double phi,
            double ref_azimuth,
            double ref_altitude,
            double cam_rot,
            double azimuth,
            double altitude,
            double focal_length,
            double * axref,
            double * ayref,
            double * phiref )
```

Transform from one camera to common reference frame.

Transform from the camera plane coordinate system of a telescope looking to altitude/azimuth to a plane coordinate system of a potential telescope looking to a reference direction ref_azimuth,ref_altitude and having unit focal length. Rotation of image angles is accounted for but not imaging errors.

References angles_to_offset(), and offset_to_angles().

Here is the call graph for this function:



7.51.2.4 get_shower_trans_matrix()

```
void get_shower_trans_matrix (
            double azimuth,
            double altitude,
            double trans[ ][3] )
```

Calculate transformation matrix.

Calculate transformation matrix from horizontal reference frame to one z axis in the given Az/Alt direction and the x axis in the plane defined by Az/Alt and zenith.

7.51.2.5 intersect_lines()

```
int intersect_lines (
            double xp1,
            double yp1,
            double phi1,
            double xp2,
            double yp2,
            double phi2,
            double * xs,
            double * ys,
            double * sang )
```

Intersect pairs of lines.

Intersect a pair of straight lines in a plane and return the intersection point and the angle at which the lines intersect.

7.51.2.6 line_point_distance()

```
double line_point_distance (
            double xp1,
            double yp1,
            double zp1,
            double cx,
            double cy,
            double cz,
            double x,
            double y,
            double z )
```

Distance between a straight line and a point in space.

Parameters

xp1,yp1,zp1	reference point on the line
cx,cy,cz	direction cosines of the line
x,y,z	point in space

Returns

distance

7.51.2.7 offset_to_angles()

```
void offset_to_angles (
            double xoff,
            double yoff,
            double azimuth,
            double altitude,
            double focal_length,
            double * obj_azimuth,
            double * obj_altitude )
```

Transform from offset in camera to corresponding Az/Alt.

Transform from the offset an object or image has in the camera plane of a telescope to the corresponding Az/Alt.

Transform from the offset an object or image has in the camera plane of a telescope to the corresponding Az/Alt.

This does not account for any rotation of the camera and its pixels. (xoff and yoff are assumed to be corrected for camera rotation). In the presence of imaging errors, an effective focal length should be used.

Referenced by cam_to_ref().

7.51.2.8 shower_geometric_reconstruction()

```
int shower_geometric_reconstruction (
            int ntel,
            const double * amp,
            const double * ximg,
            const double * yimg,
            const double * phi,
            const double * disp,
            const double * xtel,
            const double * ytel,
            const double * ztel,
            const double * az,
            const double * alt,
            const double * flen,
            const double * cam_rot,
            double ref_az,
            double ref_alt,
            int flag,
            double * shower_az,
            double * shower_alt,
            double * var_dir,
            double * xc,
            double * yc,
            double * var_core )
```

Simple reconstruction by intersecting pairs of lines.

Simple geometric shower reconstruction by intersecting pairs of straigh lines (from major axis of second moments ellipses after transformation to a common plane), first for the shower direction and then for the core position. No errors on reconstructed direction or core position are calculated. This should sooner or later be superceded by a fit procedure taking advantage of estimated errors on image positions and angles.

Parameters

ntel	The number of telescopes with suitable images.
amp	The image amplitudes in each suitable telescope [p.e.].
ximg	The image c.o.g. x positions in the local camera coordinate systems.

Parameters

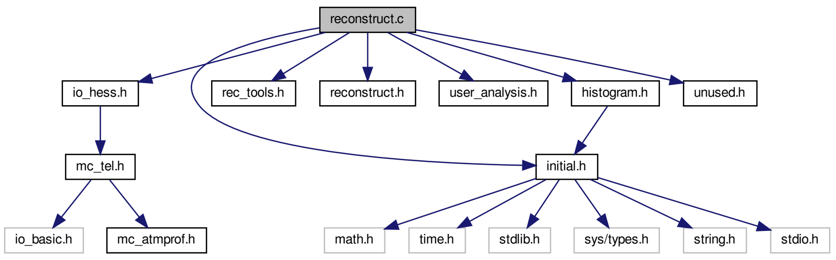
yimg	The image c.o.g. y positions in the local camera coordinate systems.
phi	The image major axis direction [rad].
disp	The DISP parameter (1.-width/length), used for giving preference to elongated images. Set all to 1.0 if unknown or no preference wanted. Can also be passed as a NULL pointer instead.
xtel	The x coordinate of the telescope positions within array [m].
ytel	The y coordinate of the telescope positions within array [m].
ztel	The z coordinate of the telescope positions within array [m].
az	The azimuth angles to which the telescopes are pointing (N->E->S->W) [rad].
alt	The altitude angles to which the telescopes are pointing [rad].
flen	The focal length to which ximg and yimg are scaled (1.0 if in units of radians, otherwise flen is in meters).
cam_rot	Camera rotation angle [rad].
ref_az	The reference azimuth angle (system nominal azimuth) [rad].
ref_alt	The reference altitude angle (system nominal altitude) [rad].
flag	Use the reconstucted direction to derive the core position (0) or use the nominal direction for that (1 or any other non-zero). The second version may sightly improve core distance and thus energy accuracy for well-defined point sources.
shower_az	Return the reconstructed shower azimuth angle (N->E->S->W) [rad].
shower_alt	Return the reconstructed shower altitude angle [rad].
var_dir	Variance (dx∗∗2+dy∗∗2)/ntel of reconstructed direction for more than two images. Can be NULL if you are not interested in it.
xc	Return the reconstructed core position x coordinate (at z=0) [m].
yc	Return the reconstructed core position y coordinate (at z=0) [m].
var_core	Variance (dx∗∗2+dy∗∗2)/ntel of reconstructed core position for more than two images. Can be NULL if you are not interested in it.

7.52 reconstruct.c File Reference

Second moments type image analysis.

```
#include "initial.h"
#include "io_hess.h"
#include "rec_tools.h"
#include "reconstruct.h"
#include "user_analysis.h"
#include "histogram.h"
#include "unused.h"
```

Include dependency graph for reconstruct.c:



Data Structures

- struct camera_nb_list

Macros

- #define CALIB_SCALE 0.92

  The factor needed to transform from mean p.e.
- #define **H_MAX_NB** 50
- #define **WITH_PZPSA** 1

Functions

- int **allocate_nb_list** (int itel, int npix, int shape_type, int nnbs, int ∗nbs)
- int **deallocate_nb_list** (int itel)
- int set_disabled_pixels (AllHessData ∗hsdata, int itel, double broken_pixels_fraction)

  Set up pixels to be ignored (regarded as zero amplitude) in the analysis if they either a) are reported as having HV disabled (no signal) in the input data stream or in a custom list, b) the camera active radius gets clipped and the pixel is outside, or c) they are randomly chosen to be ignored.
- static int guess_pixel_shape (CameraSettings ∗camset, int itel)

  Guess the common pixel shape type from relative positions of neighbours.
- static int find_neighbours (CameraSettings ∗camset, int itel)

  Find the list of neighbours for each pixel.
- int **store_camera_radius** (CameraSettings ∗camset, int itel)
- double **get_camera_radius** (int itel, int maxflag)
- void select_calibration_channel (int chn)

  Control if only low-gain or high-gain should get used instead of both.
- int calibrate_amplitude (AllHessData ∗hsdata, int itel, int flag_amp_tm, double clip_amp)

  Calibrate amplitudes in all pixels of a camera.
- double calibrate_pixel_amplitude (AllHessData ∗hsdata, int itel, int ipix, int flag_amp_tm, int itime, double clip_amp)

  Calibrate a single pixel amplitude.
- static int simple_integration (AllHessData ∗hsdata, int itel, int nsum, int nskip)

  Integrate sample-mode data (traces) over a common and fixed interval.
- static int global_peak_integration (AllHessData ∗hsdata, int itel, int nsum, int nbefore, int ∗sigamp)

  Integrate sample-mode data (traces) over a common interval around a global signal peak.
- static int local_peak_integration (AllHessData ∗hsdata, int itel, int nsum, int nbefore, int ∗sigamp)

  Integrate sample-mode data (traces) around a pixel-local signal peak.
- static int nb_peak_integration (AllHessData ∗hsdata, int lwt, int itel, int nsum, int nbefore, int ∗sigamp)

  Integrate sample-mode data (traces) around a peak in the signal sum of neighbouring pixels.
- static int **nb_peak_integration** (AllHessData ∗hsdata, int lwt, int itel, int nsum, int nbefore, _unused_ int ∗sigamp)
- static int gradient_integration (AllHessData ∗hsdata, int itel, int nsum, int nbefore, int ∗sigamp)

  Fit gradient of pixel pulse peak times along image and evaluate the fitted line for getting the time around which pulses get integrated.
- static int PzpsaSmoothUpsampleU16 (int n, int us, uint16_t ∗ip, double bl, double pz, double ∗op, double ∗max, int ∗at)

  Upsample (expand the n input values to us samples each) Subtract baseline bl and correct for a single pole decay with the decay time pz and smooth the resulting trace with two moving averages with a width of us.
- static double PzpsaPeakProperty (int n, double ∗in, int pos, int w, double ∗intsum, double ∗cog)

  Calculates the peak property of the signal in (n samples) at position pos.

- static int nb_fc_shaped_peak_integration (AllHessData ∗hsdata, int itel, int nsum, int nbefore, int ∗sigamp, int psopt, int ithr)

  Pulse integration based on peaks in neighbour pixel signals after FlashCam-style pulse shaping.
- static int nb_fc_shaped_peak_integration (AllHessData ∗hsdata, int itel, int nsum, int nbefore, _unused_ int ∗sigamp, int psopt, int ithr)
- static double qpol (double x, int np, double ∗yval)

  Quick interpolation in array of points equidistant in x coordinate.
- static int set_integration_correction (AllHessData ∗hsdata, int itel, int integrator, int ∗intpar)

  With partial pulse integration we extract a correction factor from partial to full pulse area from the reference pulse shape provided by MC.
- static int pixel_integration (AllHessData ∗hsdata, int itel, struct user_parameters ∗up)

  Pixel integration steering function.
- static int clean_image_tailcut (AllHessData ∗hsdata, int itel, double al, double ah, int lref, double minfrac)

  Use dual-level tail-cut image cleaning procedure to get pixel list.
- static int second_moments (AllHessData ∗hsdata, int itel, int cut_id, int nimg, double clip_amp)

  Reconstruction of second moments parameters from cleaned image.
- static int pixel_timing_analysis (AllHessData ∗hsdata, int itel, int nimg)

  Calculate summary results from pixel timing data.
- static int image_reconstruct (AllHessData ∗hsdata, int itel, int cut_id, double tcl, double tch, int lref, double minfrac, int nimg, int flag_amp_tm, double clip_amp)

  Calibrate and clean image pixels and reconstruct second moments parameters from images.
- int **clean_raw_data** (AllHessData ∗hsdata, int itel, int clean_flag, int tcl, int tch, struct user_parameters ∗up)
- int **clean_raw_data** (AllHessData ∗hsdata, int itel, int clean_flag, _unused_ int tcl, _unused_ int tch, _↩
unused_ struct user_parameters ∗up)
- static int fill_pixel_trg_stats (AllHessData ∗hsdata, int itel, int tel_type, struct user_parameters ∗up)
- static int shower_reconstruct (AllHessData ∗hsdata, const double ∗min_amp_tel, const size_t ∗min_pix_tel, int cut_id)

  Shower reconstruction (geometrical reconstruction only)
- int reconstruct (AllHessData ∗hsdata, int reco_flag, const double ∗min_amp, const size_t ∗min_pix, const double ∗tcl, const double ∗tch, const int ∗lref, const double ∗minfrac, int nimg, int flag_amp_tm, int clean_↩
flag)

  Image/shower reconstruction function.
- void **set_reco_verbosity** (int v)

Variables

- int **reco_verbose_level** = 0
- static int **px_shape_type** [H_MAX_TEL]
- static struct camera_nb_list nb_lists [H_MAX_TEL][3]

  To be filled with up to 3 neighbour lists for each telescope.
- static struct camera_nb_list ext_list [H_MAX_TEL]

  Optional extension lists beyond image cleaning.
- static int **image_list** [H_MAX_TEL][H_MAX_PIX]
- static int **image_numpix** [H_MAX_TEL]
- static double **pixel_amp** [H_MAX_TEL][H_MAX_PIX]
- static int **show_total_amp** = 0
- static int **pixel_sat** [H_MAX_TEL]
- static char **pixel_disabled** [H_MAX_TEL][H_MAX_PIX]
- static int **any_disabled** [H_MAX_TEL]
- static double **camera_radius_eff** [H_MAX_TEL]
- static double **camera_radius_max** [H_MAX_TEL]
- static double **integration_correction** [H_MAX_TEL][H_MAX_GAINS]
- static int **verbosity** = 0
- static int **no_low_gain** = 0
- static int **no_high_gain** = 0

7.52.1 Detailed Description

Second moments type image analysis.

Author

Konrad Bernloehr

Date

2003 to 2023

7.52.2 Macro Definition Documentation

7.52.2.1 CALIB_SCALE

```
#define CALIB_SCALE 0.92
```

The factor needed to transform from mean p.e.

units to units of the single-p.e. peak: Depends on the collection efficiency, the asymmetry of the single p.e. amplitude distribution and the electronic noise added to the signals. Default value is for HESS.

7.52.3 Function Documentation

7.52.3.1 calibrate_amplitude()

```
int calibrate_amplitude (
            AllHessData * hsdata,
            int itel,
            int flag_amp_tm,
            double clip_amp )
```

Calibrate amplitudes in all pixels of a camera.

This function is operating only on pulse sums, either from normal raw data or from timing/pulse shape analysis. Use calibrate_pixel_amplitude() for calibration of individual samples.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Index of telescope in the relevant arrays (not the ID).
flag_amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak.
clip_amp	if $>$0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

References camera_nb_list::npix, and simtel_camera_settings_struct::num_pixels.

Referenced by image_reconstruct().

7.52.3.2 calibrate_pixel_amplitude()

```
double calibrate_pixel_amplitude (
             AllHessData * hsdata,
             int itel,
             int ipix,
             int flag_amp_tm,
             int itime,
             double clip_amp )
```

Calibrate a single pixel amplitude.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Index of telescope in the relevant arrays (not the ID).
ipix	The pixel number (0 ... npix-1).
flag_amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak.
itime	-1: sum of samples of type as given in flag_amp_tm 0...(nsamples-1): sample data (if available) for one time slice
clip_amp	if >0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

Returns

Pixel amplitude in peak p.e. units (based on conversion factor from H.E.S.S.).

7.52.3.3 clean_image_tailcut()

```
static int clean_image_tailcut (
             AllHessData * hsdata,
             int itel,
             double al,
             double ah,
             int lref,
             double minfrac )  [static]
```

Use dual-level tail-cut image cleaning procedure to get pixel list.

In contrast to the classical dual-level tail-cuts this function has an optional restriction to only those pixels having an amplitude above a given fraction of the n-th hottest pixel. This should almost stop the increase of width and length with increasing intensity after some point.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Sequence number of the telescope being processed.
al	The lower of the two tail-cut thresholds.
ah	The higher of the two tail-cut thresholds.
lref	Determines which pixel, after sorting by amplitude, will be used as providing the reference amplitude. Example: use 3 for the third hottest pixel. If this number is <= 0, the classical scheme is used.
minfrac	Which fraction of the reference amplitude is required for pixels to be included in the final image. If this number is <= 0.0, the classical scheme is used.

Referenced by image_reconstruct().

7.52.3.4 fill_pixel_trg_stats()

```
static int fill_pixel_trg_stats (
             AllHessData * hsdata,
             int itel,
             int tel_type,
             struct user_parameters * up )  [static]
```

< true energy [TeV]

< Event for desired spectral slope

7.52.3.5 find_neighbours()

```
static int find_neighbours (
             CameraSettings * camset,
             int itel )  [static]
```

Find the list of neighbours for each pixel.

< Temporary neighbour lists for one telescope.

Referenced by image_reconstruct().

7.52.3.6 global_peak_integration()

```
static int global_peak_integration (
             AllHessData * hsdata,
             int itel,
             int nsum,
             int nbefore,
             int * sigamp )  [static]
```

Integrate sample-mode data (traces) over a common interval around a global signal peak.

The integration window can be anywhere in the available length of the traces. Since the calibration function subtracts a pedestal that corresponds to the total length of the traces we may also have to add a pedestal contribution for the samples not summed up. No weighting of individual samples is applied.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Sequence number of the telescope being processed.
nsum	Number of samples to sum up (is reduced if exceeding available length).
nbefore	Start the integration a number of samples before the peak, as long as it fits into the available data range. Note: for multiple gains, this results in identical integration regions.
sigamp	Amplitude in ADC counts above pedestal at which a signal is considered as significant (separate for high gain/low gain).

7.52.3.7 gradient_integration()

```
static int gradient_integration (
            AllHessData * hsdata,
            int itel,
            int nsum,
            int nbefore,
            int * sigamp )  [static]
```

Fit gradient of pixel pulse peak times along image and evaluate the fitted line for getting the time around which pulses get integrated.

There are basically three problems: a) bootstrap problem for finding significant pixels, b) robustness of the fit in case of pixels that don't follow the time gradient, and c) what to do with pixels that have a large enough signal at a time not consistent with the fitted line.

References H_MAX_TEL, simtel_tel_event_adc_struct::known, simtel_tel_event_adc_struct::num_samples, simtel_tel_event_data_struct::raw, and simtel_event_data_struct::teldata.

7.52.3.8 local_peak_integration()

```
static int local_peak_integration (
            AllHessData * hsdata,
            int itel,
            int nsum,
            int nbefore,
            int * sigamp )  [static]
```

Integrate sample-mode data (traces) around a pixel-local signal peak.

The integration window can be anywhere in the available length of the traces. Since the calibration function subtracts a pedestal that corresponds to the total length of the traces we may also have to add a pedestal contribution for the samples not summed up. No weighting of individual samples is applied.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Sequence number of the telescope being processed.
nsum	Number of samples to sum up (is reduced if exceeding available length).
nbefore	Start the integration a number of samples before the peak, as long as it fits into the available data range. Note: for multiple gains, this may result in identical integration regions (depending on signal).
sigamp	Amplitude in ADC counts above pedestal at which a signal is considered as significant (separate for high gain/low gain).

References simtel_tel_event_adc_struct::adc_known, simtel_tel_event_adc_struct::adc_sample, simtel_tel_←
event_adc_struct::adc_sum, H_MAX_TEL, HI_GAIN, simtel_tel_event_adc_struct::known, simtel_tel_event←
_adc_struct::num_gains, simtel_tel_event_adc_struct::num_pixels, simtel_tel_event_adc_struct::num_samples, simtel_tel_monitor_struct::pedestal, simtel_tel_event_data_struct::raw, simtel_tel_event_adc_struct::significant, simtel_event_data_struct::teldata, and simtel_tel_event_adc_struct::zero_sup_mode.

7.52.3.9 nb_fc_shaped_peak_integration() [1/2]

```
static int nb_fc_shaped_peak_integration (
            AllHessData * hsdata,
            int itel,
            int nsum,
            int nbefore,
            _unused_ int * sigamp,
            int psopt,
            int ithr )  [static]
```

< Pedestal in raw signal, per sample.

< Extension of summation/cog region [peakpos-w : peakpos+w]

References simtel_tel_event_adc_struct::adc_sum, simtel_pixel_timing_struct::after_peak, simtel_pixel_timing_←
struct::before_peak, H_MAX_SLICES, H_MAX_TEL, simtel_tel_event_adc_struct::known, simtel_pixel_timing←
_struct::known, simtel_pixel_timing_struct::list_size, simtel_pixel_timing_struct::list_type, simtel_tel_event_←
adc_struct::num_gains, simtel_tel_event_adc_struct::num_pixels, simtel_tel_event_adc_struct::num_samples, simtel_pixel_timing_struct::num_types, simtel_tel_event_data_struct::pixtm, simtel_tel_event_data_struct::raw, simtel_event_data_struct::teldata, simtel_pixel_timing_struct::threshold, simtel_pixel_timing_struct::time_level, and simtel_pixel_timing_struct::time_type.

7.52.3.10 nb_fc_shaped_peak_integration() [2/2]

```
static int nb_fc_shaped_peak_integration (
            AllHessData * hsdata,
            int itel,
            int nsum,
            int nbefore,
            int * sigamp,
            int psopt,
            int ithr )  [static]
```

Pulse integration based on peaks in neighbour pixel signals after FlashCam-style pulse shaping.

Basically like nb_peak_integration for lwt=0 but pulses are all upscaled in sampling frequency by a factor of four and one several variants for FlashCam-style pulse shaping is applied first. Signal extraction = integration also allows for different variants. There are actually way more variants available than necessary, intended for evaluation and testing.

Note that the psopt parameter is specified with the '–integration-window' command line option as the third value. (Recommended values for the first two are 1,0 (=nsum,nbefore). Nsum=0 means nsum=1.) Interpret psopt as decimal MHTO (with M=psopt/1000, H=(psopt%1000)/100, T=(psopt%100)/10, O=psopt%10): O = -1 : Full pzpsa shaping and peak finding over full readout range, no neighbours involved. This results in a significant bias for positive NSB fluctuations. 0 : Full pzpsa shaping but peak finding in signal of neighbours, avoiding the beginning (first 7) and end (last 3) of the upsampled signal because these are noisier and result in artifacts. (OK to use) 9 : Like '0' but include the beginning and end for peak finding. (better use 0 or 1) 1 : Like '9' but do own differencing to have smooth start and end. (recommended) 2 : Like '1' but do differencing between second-to-next original samples. 3 : Like '1' but do pulse shaping with own, more explicit code (differs in the beginning and the end but otherwise the same). 4 : Like '2' but do pulse shaping with own, more explicit code. T = 0 : Use integration from nbefore the peak for nsum upsampled samples and determine the pixel timing as the peak position close to the peak times in the signal of neighbour pixels (except for O = -1). $>$ 0 : Use the PzpsaPeakProperty code for summation and center-of-gravity determination of peaks, with T as width parameter. Nsum and nbefore are ignored. H = 0 : Summation region is entirely determined by the peak in the signal of neighbourin pixels, without any bias for NSB fluctuations. 1 : Summation region allows for small adjustement in peak position in the signal of the pixel itself. Small NSB bias. M = 0 : Not touching pixel timing structure. 1 : Re-evaluate and refill pixel timing from shaped signals and peaks. Unless a new 'integration threshold' is given, the old threshold for significant pixel timings gets re-used (but pixel is list still new).

Parameters

hsdata	Pointer to all available data and configurations.
itel	Sequence number of the telescope being processed.
nsum	Number of samples to sum up (is reduced if exceeding available length).
nbefore	Start the integration a number of samples before the peak, as long as it fits into the available data range. Note: for multiple gains, this may result in identical integration regions (depending on signal).
sigamp	(not used, just for similarity with other integration functions)
psopt	Pulse shaping option as described
ithr	Integration threshold in ADC counts gets actually used for significance in pixel timing.

Returns

0 (OK), -1 (error)

7.52.3.11 nb_peak_integration()

```
static int nb_peak_integration (
            AllHessData * hsdata,
            int lwt,
            int itel,
            int nsum,
            int nbefore,
            int * sigamp )  [static]
```

Integrate sample-mode data (traces) around a peak in the signal sum of neighbouring pixels.

The integration window can be anywhere in the available length of the traces. Since the calibration function subtracts a pedestal that corresponds to the total length of the traces we may also have to add a pedestal contribution for the samples not summed up. No weighting of individual samples is applied.

Parameters

hsdata	Pointer to all available data and configurations.
lwt	Weight of the local pixel (0: peak from neighbours only, 1: local pixel counts as much as any neighbour).

Parameters

itel	Sequence number of the telescope being processed.
nsum	Number of samples to sum up (is reduced if exceeding available length).
nbefore	Start the integration a number of samples before the peak, as long as it fits into the available data range. Note: for multiple gains, this results in identical integration regions.
sigamp	Amplitude in ADC counts above pedestal at which a signal is considered as significant (separate for high gain/low gain).

7.52.3.12 pixel_integration()

```
static int pixel_integration (
            AllHessData * hsdata,
            int itel,
            struct user_parameters * up )  [static]
```

Pixel integration steering function.

Work is done in selected integration function.

7.52.3.13 PzpsaPeakProperty()

```
static double PzpsaPeakProperty (
            int n,
            double * in,
            int pos,
            int w,
            double * intsum,
            double * cog )  [static]
```

Calculates the peak property of the signal in (n samples) at position pos.

The signal is integrated from sample pos-w to pos+w and the result is stored in intsum.

The cog is the center of gravity calculated by the area above the minumum of the signal from pos-w to pos+w

Returns a quality value for the signal which is defined as in[pos]-(in[start]+in[stop])/2. Negativ values indicate that no positive signal was found.

7.52.3.14 PzpsaSmoothUpsampleU16()

```
static int PzpsaSmoothUpsampleU16 (
            int n,
            int us,
            uint16_t * ip,
            double bl,
            double pz,
            double * op,
            double * max,
            int * at )  [static]
```

Upsample (expand the n input values to us samples each) Subtract baseline bl and correct for a single pole decay with the decay time pz and smooth the resulting trace with two moving averages with a width of us.

The output is placed in array op and returns the new number of samples (n∗us).

This function derived from code by T.Kihm, using uint16_t for input array element type and double for output. Example: PzpsaSmoothUpsampleU16(50,4,tti,0.,mpz,tto,&mxop,&imxop);

Parameters

n	Number of elements in input array ip
us	Upsampling factor (use '4' to upsample from 250 MHZ to one GHz).
ip	Pointer to input array of ADC raw data of type uint16_t
bl	Baseline (pedestal) on input per sample
pz	Pole-zero compensation factor in differencing (0<=pz<=1)
op	Pointer to output array of type double
max	Maximum content in output array (only filled if not NULL)
at	Position of maximum bin in output array (only filled if not NULL)

< running indices

< the next and prev. input samples

< the running sum of 1.st and 2.nd average

< a temp var for intermediate copy

< the next and prev. pz corrected value

< the out pointer of the first runsum

< the out pointer of the second runsum

< the multiplier to correct the two runsums

< peak maximum

< peak position

7.52.3.15 reconstruct()

```
int reconstruct (
            AllHessData * hsdata,
            int reco_flag,
            const double * min_amp,
            const size_t * min_pix,
            const double * tcl,
            const double * tch,
            const int * lref,
            const double * minfrac,
            int nimg,
            int flag_amp_tm,
            int clean_flag )
```

Image/shower reconstruction function.

Parameters

hsdata	Pointer to all available data and configurations.
reco_flag	If >= 3 then redo image cleaning before shower reconstruction. If >= 4 then the total image intensities are re-determined and that may change which images are used or not in the shower reconstruction.

Parameters

min_amp	The minimum amplitude required in images (telescope-specific, that means requiring an array of at least size H_MAX_TEL).
min_pix	The minimum number of pixels required in images (telescope-specific).
tcl	The lower of the two tail-cut thresholds (telescope-specific).
tch	The higher of the two tail-cut thresholds (telescope-specific).
lref	Determines which pixel, after sorting by amplitude, will be used as providing the reference amplitude (telescope-specific). Example: use 3 for the third hottest pixel. If this number is <= 0, the classical scheme is used.
minfrac	Which fraction of the reference amplitude is required for pixels to be included in the final image (telescope-specific). If this number is <= 0.0, the classical scheme is used.
nimg	Which of (sometimes) several images should be filled? Use -1 to replace an existing image of the same cut id (if such an image exists) or add another image (if there is free space for it) or replace the first image (if all else fails). Use -2 to indicate that image analysis from normal integrated amplitude should go into first image and (if available) that from pixel timing (around local peak position or otherwise global peak position) should go into the second image.
flag_amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak.

7.52.3.16 select_calibration_channel()

```
void select_calibration_channel (
             int chn )
```

Control if only low-gain or high-gain should get used instead of both.

Parameters

chn	0 (both channels), 1 (only high gain), 2 (only low gain)

7.52.3.17 set_disabled_pixels()

```
int set_disabled_pixels (
             AllHessData * hsdata,
             int itel,
             double broken_pixels_fraction )
```

Set up pixels to be ignored (regarded as zero amplitude) in the analysis if they either a) are reported as having HV disabled (no signal) in the input data stream or in a custom list, b) the camera active radius gets clipped and the pixel is outside, or c) they are randomly chosen to be ignored.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Telescope index where we set new values.
broken_pixels_fraction	Optional fraction of additional pixels to be set like dead pixels (not usable for analysis).

Disabled pixels are ignored in the evaluation of the camera radius.

References camera_nb_list::npix.

7.52.3.18 set_integration_correction()

```
static int set_integration_correction (
             AllHessData * hsdata,
             int itel,
             int integrator,
             int * intpar )  [static]
```

With partial pulse integration we extract a correction factor from partial to full pulse area from the reference pulse shape provided by MC.

Since actual pulses may have an intrinsic width (and as a result are wider than the reference pulse) this can still lead to a bit underestimated p.e. values. But this is hard to fix without knowing the true width of light pulses.

References H_MAX_TEL, simtel_camera_organisation_struct::num_gains, simtel_tel_event_adc_struct::num↩_samples, simtel_tel_event_data_struct::raw, simtel_pixel_setting_struct::ref_step, simtel_event_data_struct↩::teldata, and simtel_pixel_setting_struct::time_slice.

7.52.3.19 simple_integration()

```
static int simple_integration (
             AllHessData * hsdata,
             int itel,
             int nsum,
             int nskip )  [static]
```

Integrate sample-mode data (traces) over a common and fixed interval.

The integration window can be anywhere in the available length of the traces. Since the calibration function subtracts a pedestal that corresponds to the total length of the traces we may also have to add a pedestal contribution for the samples not summed up. No weighting of individual samples is applied.

Parameters

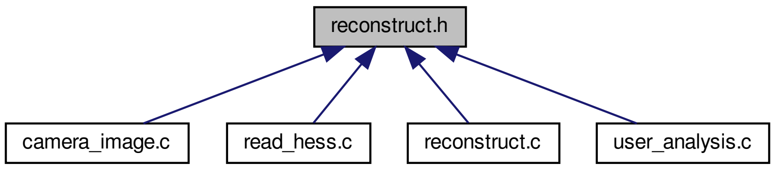
hsdata	Pointer to all available data and configurations.
itel	Sequence number of the telescope being processed.
nsum	Number of samples to sum up (is reduced if exceeding available length).
nskip	Number of initial samples skipped (adapted such that interval fits into what is available). Note: for multiple gains, this results in identical integration regions.

References simtel_tel_event_adc_struct::adc_known, simtel_tel_event_adc_struct::adc_sample, simtel_tel_↩event_adc_struct::adc_sum, H_MAX_TEL, simtel_tel_event_adc_struct::known, simtel_tel_event_adc_struct↩::num_gains, simtel_tel_event_adc_struct::num_pixels, simtel_tel_event_adc_struct::num_samples, simtel_tel↩_monitor_struct::pedestal, simtel_tel_event_data_struct::raw, simtel_tel_event_adc_struct::significant, simtel_↩event_data_struct::teldata, and simtel_tel_event_adc_struct::zero_sup_mode.

7.53 reconstruct.h File Reference

Function prototypes for reconstruct.c.

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



Functions

- int **deallocate_nb_list** (int itel)
- double line_point_distance (double xp1, double yp1, double zp1, double cx, double cy, double cz, double x, double y, double z)

    Distance between a straight line and a point in space.
- int reconstruct (AllHessData ∗hsdata, int reco_flag, const double ∗min_amp, const size_t ∗min_pix, const double ∗tcl, const double ∗tch, const int ∗lref, const double ∗minfrac, int nimg, int flag_amp_tm, int clean_↩ flag)

    Image/shower reconstruction function.
- int **store_camera_radius** (CameraSettings ∗camset, int itel)
- double **get_camera_radius** (int itel, int maxflag)
- void select_calibration_channel (int chn)

    Control if only low-gain or high-gain should get used instead of both.
- int calibrate_amplitude (AllHessData ∗hsdata, int itel, int flag_amp_tm, double clip_amp)

    Calibrate amplitudes in all pixels of a camera.
- double calibrate_pixel_amplitude (AllHessData ∗hsdata, int itel, int ipix, int flag_amp_tm, int itime, double clip_amp)

    Calibrate a single pixel amplitude.
- double **calibrate_pixel_sample_amplitude** (AllHessData ∗hsdata, int itel, int ipix, int flag_amp_tm, int itime, double clip_sample_amp)
- void **set_reco_verbosity** (int v)
- int set_disabled_pixels (AllHessData ∗hsdata, int itel, double broken_pixels_fraction)

    Set up pixels to be ignored (regarded as zero amplitude) in the analysis if they either a) are reported as having HV disabled (no signal) in the input data stream or in a custom list, b) the camera active radius gets clipped and the pixel is outside, or c) they are randomly chosen to be ignored.

Variables

- int **reco_verbose_level**

7.53.1 Detailed Description

Function prototypes for reconstruct.c.

Author

Konrad Bernloehr

Date

2006 to 2022

7.53.2 Function Documentation

7.53.2.1 calibrate_amplitude()

```
int calibrate_amplitude (
            AllHessData * hsdata,
            int itel,
            int flag_amp_tm,
            double clip_amp )
```

Calibrate amplitudes in all pixels of a camera.

This function is operating only on pulse sums, either from normal raw data or from timing/pulse shape analysis. Use calibrate_pixel_amplitude() for calibration of individual samples.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Index of telescope in the relevant arrays (not the ID).
flag_amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak.
clip_amp	if $>$0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

References camera_nb_list::npix, and simtel_camera_settings_struct::num_pixels.

Referenced by image_reconstruct().

7.53.2.2 calibrate_pixel_amplitude()

```
double calibrate_pixel_amplitude (
            AllHessData * hsdata,
            int itel,
            int ipix,
            int flag_amp_tm,
            int itime,
            double clip_amp )
```

Calibrate a single pixel amplitude.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Index of telescope in the relevant arrays (not the ID).
ipix	The pixel number (0 ... npix-1).
flag_amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak.
itime	-1: sum of samples of type as given in flag_amp_tm 0...(nsamples-1): sample data (if available) for one time slice
clip_amp	if >0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

Returns

Pixel amplitude in peak p.e. units (based on conversion factor from H.E.S.S.).

7.53.2.3 line_point_distance()

```
double line_point_distance (
              double xp1,
              double yp1,
              double zp1,
              double cx,
              double cy,
              double cz,
              double x,
              double y,
              double z )
```

Distance between a straight line and a point in space.

Parameters

xp1,yp1,zp1	reference point on the line
cx,cy,cz	direction cosines of the line
x,y,z	point in space

Returns

distance

Referenced by mc_event_fill().

7.53.2.4 reconstruct()

```
int reconstruct (
            AllHessData * hsdata,
            int reco_flag,
            const double * min_amp,
            const size_t * min_pix,
            const double * tcl,
            const double * tch,
            const int * lref,
            const double * minfrac,
            int nimg,
            int flag_amp_tm,
            int clean_flag )
```

Image/shower reconstruction function.

Parameters

hsdata	Pointer to all available data and configurations.
reco_flag	If >= 3 then redo image cleaning before shower reconstruction. If >= 4 then the total image intensities are re-determined and that may change which images are used or not in the shower reconstruction.
min_amp	The minimum amplitude required in images (telescope-specific, that means requiring an array of at least size H_MAX_TEL).
min_pix	The minimum number of pixels required in images (telescope-specific).
tcl	The lower of the two tail-cut thresholds (telescope-specific).
tch	The higher of the two tail-cut thresholds (telescope-specific).
lref	Determines which pixel, after sorting by amplitude, will be used as providing the reference amplitude (telescope-specific). Example: use 3 for the third hottest pixel. If this number is <= 0, the classical scheme is used.
minfrac	Which fraction of the reference amplitude is required for pixels to be included in the final image (telescope-specific). If this number is <= 0.0, the classical scheme is used.
nimg	Which of (sometimes) several images should be filled? Use -1 to replace an existing image of the same cut id (if such an image exists) or add another image (if there is free space for it) or replace the first image (if all else fails). Use -2 to indicate that image analysis from normal integrated amplitude should go into first image and (if available) that from pixel timing (around local peak position or otherwise global peak position) should go into the second image.
flag_amp_tm	0: Use normal integrated amplitude. 1: Use integration around global peak position from pulse shape analysis. May include all pixels or only selected. 2: Use integration around local peak position from pulse shape analysis. Return 0 for pixels without a fairly significant peak.

7.53.2.5 select_calibration_channel()

```
void select_calibration_channel (
            int chn )
```

Control if only low-gain or high-gain should get used instead of both.

Parameters

chn	0 (both channels), 1 (only high gain), 2 (only low gain)

7.53.2.6 set_disabled_pixels()

```
int set_disabled_pixels (
            AllHessData * hsdata,
            int itel,
            double broken_pixels_fraction )
```

Set up pixels to be ignored (regarded as zero amplitude) in the analysis if they either a) are reported as having HV disabled (no signal) in the input data stream or in a custom list, b) the camera active radius gets clipped and the pixel is outside, or c) they are randomly chosen to be ignored.

Parameters

hsdata	Pointer to all available data and configurations.
itel	Telescope index where we set new values.
broken_pixels_fraction	Optional fraction of additional pixels to be set like dead pixels (not usable for analysis).

Disabled pixels are ignored in the evaluation of the camera radius.

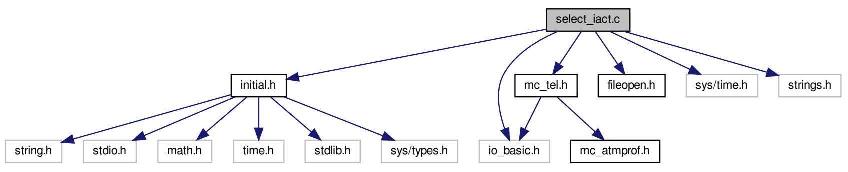
References camera_nb_list::npix.

7.54 select_iact.c File Reference

A program reading simulated CORSIKA data written through the IACT interface and, if it contains extra information on particles emitting Cherenkov light, reduce to light from selected particles.

```
#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 select_iact.c:



Data Structures

- struct selector

Macros

- #define **MAXTEL** 5

Typedefs

- typedef struct selector **Selector**

Functions

- int array_select_mc_phot (IO_BUFFER ∗iobuf)

    Select Monte Carlo photons.
- int **tel_select_mc_phot** (IO_BUFFER ∗iobuf)
- int **tel_select_mc_phot3d** (IO_BUFFER ∗iobuf)
- void **add_selector** (double m1, double m2, double E1, double E2, int c)
- int **select_bunches** (struct bunch ∗bunches, int ∗nbunches, double ∗photons)
- int **select_bunches3d** (struct bunch3d ∗bunches, int ∗nbunches, double ∗photons)
- void **ioerrorcheck** (void)
- void **syntax** (void)
- int main (int argc, char ∗∗argv)

    Main program.

Variables

- struct bunch ∗ **tel_bunches** [MAXTEL]
- struct bunch3d ∗ **tel_bunches3d** [MAXTEL]
- int **max_bunches** [MAXTEL]
- int **max_bunches3d** [MAXTEL]
- int **tel_nbunches** [MAXTEL]
- int **tel_nbunches3d** [MAXTEL]
- double **tel_photons** [MAXTEL]
- double **tel_photons3d** [MAXTEL]
- Selector ∗ **selectors** = NULL
- size_t **nselect** = 0
- static int **verbose** = 0
- struct bunch ∗ **sel_bunch** = NULL
- struct bunch3d ∗ **sel_bunch3d** = NULL
- int **sel_max** = 0
- int **sel_max3d** = 0

7.54.1 Detailed Description

A program reading simulated CORSIKA data written through the IACT interface and, if it contains extra information on particles emitting Cherenkov light, reduce to light from selected particles.

Relevant environment variables: PRINT_TEL_VERBOSE MAX_PRINT_ARRAY

Author

Konrad Bernloehr

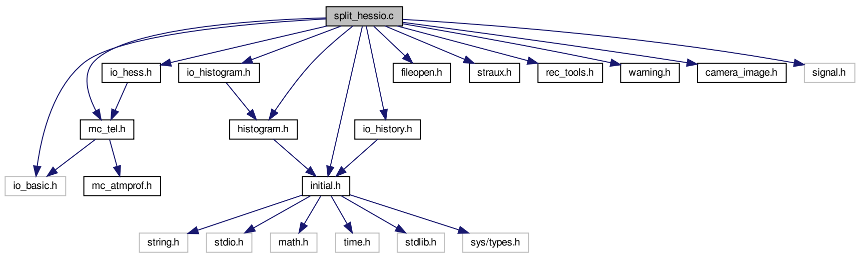
Date

2021

7.55 split_hessio.c File Reference

Rip out data for each telescope into individual files.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_history.h"
#include "io_hess.h"
#include "histogram.h"
#include "io_histogram.h"
#include "fileopen.h"
#include "straux.h"
#include "rec_tools.h"
#include "warning.h"
#include "camera_image.h"
#include <signal.h>
```
Include dependency graph for split_hessio.c:



Functions

- void stop_signal_function (int isig)

  Stop the program gracefully when it catches an INT or TERM signal.
- static void syntax (char ∗program)

  Show program syntax.
- int main (int argc, char ∗∗argv)

  Main program.

Variables

- static int **interrupted**

7.55.1 Detailed Description

Rip out data for each telescope into individual files.

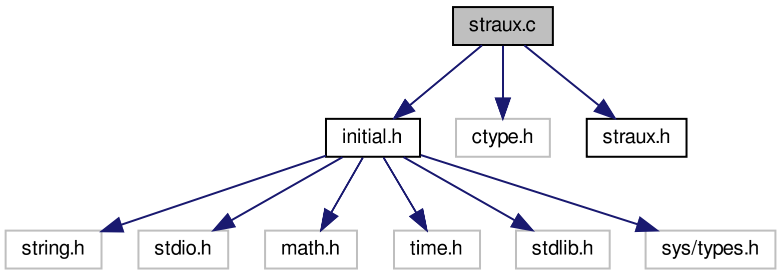
```
Rip out data for each telescope into individual files.

Syntax: split_hessio [ options ] [ - | input_fname ... ]
Options:
   -x               (Extract TelescopeEvent data from Event.)
   -X               (Extract TelescopeEvent raw data (samples or sum).)
   -i|--ignore      (Ignore unknown data block types.)
   -q|--quiet       (More quiet on standard output.)
   -v|--verbose     (More verbose on standard output.)
   --max-events n   (Skip remaining data after so many triggered events.)
   --pure-raw       (Discard any sub-items of TelescopeEvent which are not raw data.)
   --clean-history (Drop previous history data blocks)
   --output-path d (Create output files in given directory instead of current.)
   --only-telescope[s] (Only data for the given telescopes IDs is written.)
   --not-telescope[s]  (No data for the given telescopes IDs is written.)


@author  Konrad Bernloehr
@date    2014 to 2022
```

7.56 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 > delimeted 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.

7.56.1 Detailed Description

Check for abbreviations of strings and get words from strings.

Author

Konrad Bernloehr

Date

2001 to 2018

7.56.2 Function Documentation

7.56.2.1 abbrev()

```
int abbrev (
            CONST char * s,
            CONST char * t )
```

Compare strings s and 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

s	The string to be checked.
t	The test string with minimum part in upper case.

Returns

1 if s is an abbreviation of t, 0 if not.

7.56.2.2 getword()

```
int getword (
            CONST char * s,
            int * spos,
            char * word,
            int maxlen,
            char blank,
            char endchar )
```

Copies a blank or '\0' or < endchar > delimeted 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.

The word must have a length less than or equal to maxlen.

Parameters

s	string with any number of words.
spos	position in the string where we start and end.
word	the extracted word.
maxlen	the maximum allowed length of word.
blank	has the same effect as ' ', i.e. end-of-word.
endchar	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

Referenced by push_config_history(), and user_set_tel_type_param_by_str().

7.56.2.3 stricmp()

```
int stricmp (
            CONST char * a,
            CONST char * b )
```

Case independent comparison of character strings.

Parameters

a,b	– strings to be compared.

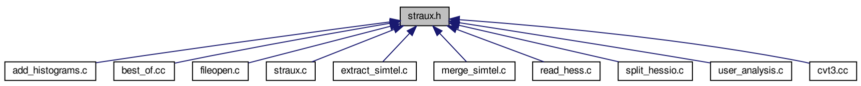
Returns

0 : strings are equal (except perhaps for case) >0 : a is lexically 'greater' than b <0 : a is lexically 'smaller' than b

7.57 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 > delimeted 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.

7.57.1 Detailed Description

Check for abbreviations of strings and get words from strings.

Author

Konrad Bernloehr

Date

2001 to 2018

7.57.2 Function Documentation

7.57.2.1 abbrev()

```
int abbrev (
            CONST char * s,
            CONST char * t )
```

Compare strings s and 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

s	The string to be checked.
t	The test string with minimum part in upper case.

Returns

1 if s is an abbreviation of t, 0 if not.

7.57.2.2 getword()

```
int getword (
            CONST char * s,
            int * spos,
            char * word,
            int maxlen,
            char blank,
            char endchar )
```

Copies a blank or '\0' or $<$ endchar $>$ delimeted 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.

The word must have a length less than or equal to maxlen.

Parameters

s	string with any number of words.
spos	position in the string where we start and end.
word	the extracted word.
maxlen	the maximum allowed length of word.
blank	has the same effect as ' ', i.e. end-of-word.
endchar	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

7.57.2.3 stricmp()

```
int stricmp (
            CONST char * a,
            CONST char * b )
```

Case independent comparison of character strings.

Parameters

a,b	– strings to be compared.

Returns

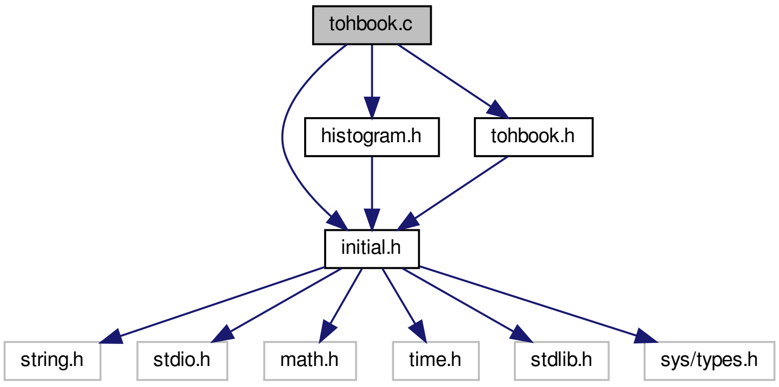
0 : strings are equal (except perhaps for case) $>$0 : a is lexically 'greater' than b $<$0 : a is lexically 'smaller' than b

7.58 tohbook.c File Reference

Convert my histograms to HBOOK (PAW) histograms.

```
#include "initial.h"
#include "histogram.h"
#include "tohbook.h"
```

Include dependency graph for tohbook.c:



Functions

- void **convert_histograms_to_hbook** (const char ∗fname)
- int **histogram_to_hbook** (int ihisto, HISTOGRAM ∗histo)

7.58.1 Detailed Description

Convert my histograms to HBOOK (PAW) histograms.

Author

Konrad Bernloehr

Date

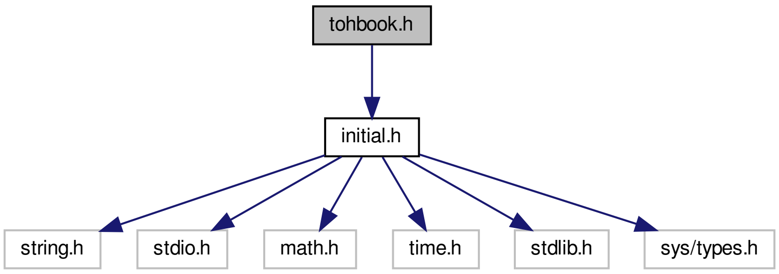
2001 to 2018

7.59 tohbook.h File Reference

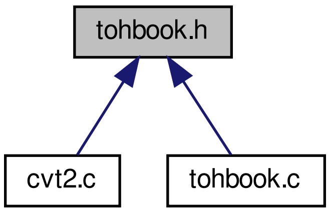
Macros and function declarations to call CERN Library HBOOK functions.

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

Include dependency graph for tohbook.h:



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



Macros

- #define **BEGIN_HBOOK**() beginhbook_()
- #define **SAVE_HBOOK**(TITLE, ID) savehbook_(TITLE,&ID,strlen(TITLE))
- #define **HBOOK1**(ID, CHTITLE, NX, XMI, XMA, VMX)
- #define **HBOOK2**(ID, CHTITLE, NX, XMI, XMA, NY, YMI, YMA, VMX)
- #define **HPAK**(ID, DATA) do { int _arg_ID=ID; hpak_(&_arg_ID,DATA); } while(0)
- #define **HFILL**(ID, X, Y, WEIGHT)
- #define **HEXIST**(ID) hexist_call(ID)

Functions

- void **beginhbook_** (void)
- void **savehbook_** (const char ∗, int ∗, size_t)
- void **hbook1_** (int ∗, const char ∗, int ∗, float ∗, float ∗, float ∗, size_t)
- void **hbook2_** (int ∗, const char ∗, int ∗, float ∗, float ∗, int ∗, float ∗, float ∗, float ∗, size_t)
- void **hpak_** (int ∗, float ∗)
- void **hfill_** (int ∗, float ∗, float ∗, float ∗)
- int **hexist_** (int ∗)
- static int **hexist_call** (int id)
- void **convert_histograms_to_hbook** (const char ∗fname)
- int **histogram_to_hbook** (int ihisto, HISTOGRAM ∗histo)

7.59.1 Detailed Description

Macros and function declarations to call CERN Library HBOOK functions.

Author

Konrad Bernloehr

Date

1992 to 2020

7.59.2 Macro Definition Documentation

7.59.2.1 HBOOK1

```
#define HBOOK1(
                ID,
                CHTITLE,
                NX,
                XMI,
                XMA,
                VMX )
```

Value:

```
  do { int _arg_ID=ID, _arg_NX=NX; \
       float _arg_XMI=XMI, _arg_XMA=XMA, _arg_VMX=VMX; \
       hbook1_(&_arg_ID,CHTITLE,&_arg_NX,&_arg_XMI,&_arg_XMA,&_arg_VMX, \
       strlen(CHTITLE)); } while(0)
```

7.59.2.2 HBOOK2

```
#define HBOOK2(
                ID,
                CHTITLE,
                NX,
                XMI,
                XMA,
                NY,
                YMI,
                YMA,
                VMX )
```

Value:

```
do { int _arg_ID=ID, _arg_NX=NX, _arg_NY=NY; \
     float _arg_XMI=XMI, _arg_XMA=XMA, \
       _arg_YMI=YMI, _arg_YMA=YMA, _arg_VMX=VMX; \
     hbook2_(&_arg_ID,CHTITLE,&_arg_NX,&_arg_XMI,&_arg_XMA, \
       &_arg_NY,&_arg_YMI,&_arg_YMA,&_arg_VMX, \
     strlen(CHTITLE)); } while(0)
```

7.59.2.3 HFILL

```
#define HFILL(
                ID,
                X,
                Y,
                WEIGHT )
```

Value:

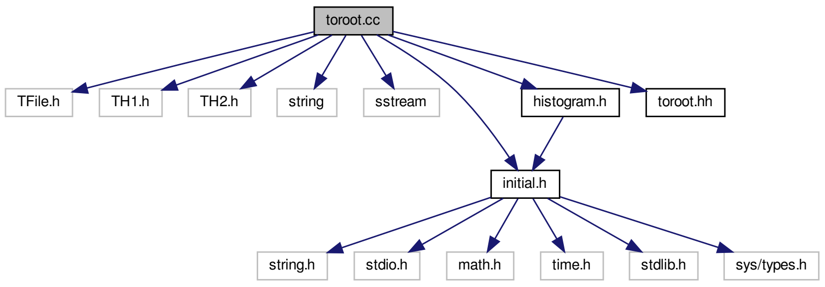
```
do { int _arg_ID=ID; float _arg_X=X, _arg_Y=Y, _arg_WEIGHT=WEIGHT; \
   hfill_(&_arg_ID,&_arg_X,&_arg_Y,&_arg_WEIGHT); } while(0)
```

7.60 toroot.cc File Reference

Functions for conversion of eventio histograms to ROOT format.

```
#include <TFile.h>
#include <TH1.h>
#include <TH2.h>
#include <string>
#include <sstream>
#include "initial.h"
#include "histogram.h"
```

```
#include "toroot.hh"
```
Include dependency graph for toroot.cc:



Functions

- string num2str (int i)

  Convert an int to a string using the STL.
- string num2str (double d)

  Convert a double to a string using the STL.
- template<class T >
  string num2str (T num)

  Convert various sorts of numbers to a string.
- void convert_histograms_to_root (const char ∗fname)

  Open a ROOT file for output, convert all histograms known and write to file.
- int histogram_to_root (int ihisto, HISTOGRAM ∗histo)

  Create a ROOT histogram from the eventio histogram.

7.60.1 Detailed Description

Functions for conversion of eventio histograms to ROOT format.

Author

Konrad Bernloehr

Date

2002 to 2018

7.60.2 Function Documentation

7.60.2.1 convert_histograms_to_root()

```
void convert_histograms_to_root (
            const char * fname )
```

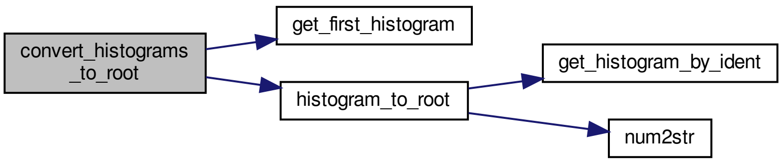
Open a ROOT file for output, convert all histograms known and write to file.

Parameters

fname	Name of ROOT output file.

References get_first_histogram(), histogram_to_root(), and histogram::next.

Here is the call graph for this function:



7.60.2.2 histogram_to_root()

```
int histogram_to_root (
             int ihisto,
             HISTOGRAM * histo )
```

Create a ROOT histogram from the eventio histogram.

Create a ROOT histogram and fill it with the contents of the given histogram, if it contains any entries. If the histogram has an ID number, it is booked with this Id. Otherwise, 90000 + a sequential number is used.

Parameters

ihisto	Histogram sequential number
histo	Histogram pointer

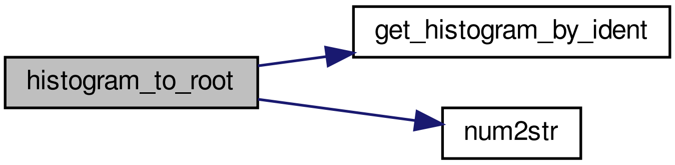
Returns

0 (ok), -1 (invalid histogram)

References Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, histogram↩ ::entries, histogram::extension, Histogram_Extension::fdata, get_histogram_by_ident(), histogram::ident, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, num2str(), histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, histogram::title, histogram↩ ::type, histogram::underflow, histogram::underflow_2d, and Histogram_Parameters::upper_limit.

Referenced by convert_histograms_to_root().

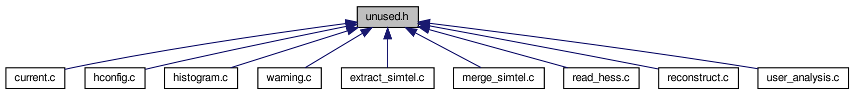
Here is the call graph for this function:



7.61 unused.h File Reference

Pre-processor macro definitions used to tell the compiler/user/documentation that a function parameter may be or definitely is unused.

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



7.61.1 Detailed Description

Pre-processor macro definitions used to tell the compiler/user/documentation that a function parameter may be or definitely is unused.

No point in warning about it. Compiler-dependent.

Author

Konrad Bernloehr

Date

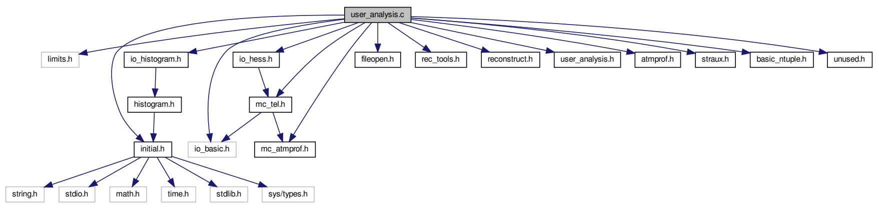
2023

7.62 user_analysis.c File Reference

Code for analysis of simulated (and reconstructed) showers within the framework of the read_hess program.

```
#include <limits.h>
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "io_hess.h"
#include "io_histogram.h"
#include "fileopen.h"
#include "rec_tools.h"
#include "reconstruct.h"
#include "user_analysis.h"
#include "mc_atmprof.h"
#include "atmprof.h"
#include "straux.h"
#include "basic_ntuple.h"
#include "unused.h"
```

Include dependency graph for user_analysis.c:



Data Structures

- struct tel_type_param
- struct telescope_list
- struct ebias_cor_data

Macros

- #define **MAX_TEL_TYPES** 10
- #define **PATH_MAX** 4096

Functions

- static void interp (double x, double ∗v, int n, int ∗ipl, double ∗rpl)

  Linear interpolation with binary search algorithm.
- static double rpol (double ∗x, double ∗y, int n, double xp)

  Linear interpolation with binary search algorithm.
- void user_set_lookup_file (const char ∗fname)

  Override the automatic naming for lookup files.
- void user_set_histogram_file (const char ∗fname)

  Override the automatic naming for histogram files.

- void user_set_telescope_type (int itype)

  Select a specific telescope type for setting user parameters.
- int user_set_tel_type_param_by_str (const char ∗str)

  Set telescope type parameters from a string (e.g.
- int which_telescope_type (const CameraSettings ∗cam_set)

  Find out to which telescope type a telescope belongs, by best matching in the required parameters.
- struct user_parameters ∗ **user_get_parameters** (int tp)
- int user_get_type (int itel)

  Get the best matching telescope type for a given telescope index.
- static double eval_cut_param (double ∗cut, double lgE)

  Evaluate energy-dependent cut parameters with.
- void **__attribute__** ((constructor))
- void user_set_flags (int uf)

  Set user-defined flags: used to active HESS-style analysis.
- void user_set_spectrum (double di)

  Set the difference between generated MC spectrum and the assumed source spectrum.
- void user_set_impact_range (double ∗impact_range)

  Set the acceptable ranges for reconstructed impact positions.
- void user_set_true_impact_range (double ∗true_impact_range)

  Set the acceptable ranges for true impact positions.
- void user_set_max_core_distance (double rt)

  Set the maximum core distance for telescopes if their images should be used beyond geometrical reconstruction.
- void user_set_min_amp (double a)

  Set the minimum amplitude of images usable for the analysis.
- void user_set_tail_cuts (double tcl, double tch, int lref, double minfrac)

  Set the lower and upper tail cuts for the standard two-level tail-cut scheme.
- void user_set_min_pix (int mpx)

  Set the minimum number of significant pixels in usable images.
- void user_set_reco_flag (int rf)

  Set the reconstruction level flag ('-r' option in read_hess).
- void user_set_tel_img (int tmn, int tmx)

  Set the minimum and maximum number of usable images for events used in analysis.
- void user_set_tel_list (size_t min_tel, size_t ntel, int ∗tel_id)

  You may have alternative selections of (fewer) telescopes.
- void user_set_max_theta (double thmax, double thscale, double thmin)

  Set the maximum angle between source and reconstructed shower direction.
- void user_set_theta_escale (double ∗thes)

  By default the angular acceptance is the 80% containment radius.
- void user_set_de_cut (double ∗dec)

  The dE cut can be made more or less strict by a scale parameter which should be 1.0 by default and is below 1 for a stricter cut and above 1 for a looser cut.
- void user_set_de2_cut (double ∗de2c)

  Since the dE2 cut is not always of any help with default cut parameters, you can change the parameter to your needs.
- void user_set_hmax_cut (double hmaxc)

  The hmax cut can be made or or less strict by a scale parameter which should be 1.0 by default and is below 1 for a stricter cut and above 1 for a looser cut.
- void user_set_shape_cuts (double wmin, double wmax, double lmin, double lmax)

  Set shape cut parameters.
- void user_set_width_max_cut (double ∗wmax)

  Set energy dependent scaled width limit.
- void user_set_length_max_cut (double ∗lmax)

Set energy dependent scaled length limit.
- void user_set_focal_length (double f)

  Set the telescope effective focal length.
- void user_set_clipping (double dc)

  Set the maximum radius to be used of a camera.
- void user_set_clipamp (double cpa)

  Set the maximum amplitude in a pixel.
- void user_set_trg_req (int trg_req)

  Set the required trigger type(s) as a bit pattern.
- void **user_set_diffuse_mode** (int dm, double oar[ ])
- void **user_set_verbosity** (int v)
- int **user_selected_event** ()
- void **user_set_auto_lookup** (int al)
- void **user_set_integrator** (int scheme)
- void **user_set_integ_window** (int nsum, int noff, int ps_opt)
- void **user_set_integ_threshold** (int ithg, int itlg)
- void **user_set_integ_no_rescale** (int no)
- void **user_set_calib_scale** (double s)
- void **user_set_nb_radius** (double ∗r)
- void **user_set_nxt_radius** (double r)
- void **user_set_pixel_stats** (int on)
- static double expected_max_height (double E, double theta, double height)

  Expected height of the shower maximum above the detector for gamma rays, based on simple analytical formula and exponential atmospheric profile.
- static double expected_max_distance (double E, double theta, double height)

  Expected distance of the shower maximum from the detector for gamma rays, based on simple analytical formula and exponential atmospheric profile.
- static int img_norm (double w, double l, double A, double lgA, double rc, int tel_type, double ∗scrw, double ∗scrl, double ∗scw, double ∗scl, double ∗sce, double ∗scer, double ∗rco, double ∗rcor, double ∗dimgo, double ∗dimgor)

  Get scaled + reduced scaled image parameters (both HEGRA and HESS type scaling) as well as energy scaling from the lookups.
- double ebias_correction (double lgE)

  Ask for a correction to log10(reconstructed energy), if available.
- void set_ebias_correction (HISTOGRAM ∗h)

  Set correction to log10(reconstructed energy), if available.
- static void init_telescope_types (AllHessData ∗hsdata)

  Initialize what of type each telescope is.
- static void **book_hist_global** (AllHessData ∗hsdata)
- static void **book_hist_for_type** (AllHessData ∗hsdata, int itype)
- static void **book_hist_for_type** (_unused_ AllHessData ∗hsdata, int tel_type)
- static void user_init (AllHessData ∗hsdata)

  Initialisation of user analysis, booking of histograms etc.
- static void user_mc_shower_fill (_unused_ AllHessData ∗hsdata)

  Work to be done once per generated shower.
- static void user_mc_event_fill (AllHessData ∗hsdata)

  Work to be done once per shower usage.
- static void user_event_fill (AllHessData ∗hsdata, int stage)

  Fill (triggered) event specific histograms etc.
- static void user_done (_unused_ AllHessData ∗hsdata)

  After all data for a file (usually one run) was processed.
- static char ∗ prog_path (void)

  Find the path from which the current program was started.
- static void user_finish (AllHessData ∗hsdata)

  Final call before program terminates.
- int **do_user_ana** (AllHessData ∗hsdata, unsigned long item_type, int stage)

Variables

- static int **verbosity** = 0
- static int **user_init_done** = 0
- static int **current_tel_type** = 0
- static struct tel_type_param **def_tel_type_param** [MAX_TEL_TYPES]
- static int **saved_tel_type** [H_MAX_TEL]
- static char **user_lookup_fname** [2048]
- static char **hist_fname** [2048]
- static struct telescope_list ∗ **alt_list** = NULL
- static size_t **n_list** = 0
- static double **max_theta** = 0.2 ∗ (M_PI/180.)
- static double **min_theta** = 0.2 ∗ (M_PI/180.)
- static struct user_parameters **up** [MAX_TEL_TYPES+2]
- static int nparams

  Number of parameters, including: the gamma-ray source offset plus d_sp_idx, min_amp, tailcut_low, tailcut_high, min_pix, reco_flag, min_tel_img, max_tel_img, max_theta, theta_scale.
- static int **nparams_i**
- static int **nparams_d**
- static double ∗ **params**
- static double opt_theta_cut [7][H_MAX_TEL]

  Angular cut limit is multiplicity dependent.
- static int **diffuse_mode** = 0
- static double **diffuse_off_axis_min** = 0.
- static double **diffuse_off_axis_max** = M_PI/2.
- static int **event_selected** = 0
- static int **auto_lookup** = 0
- static int telescope_type [H_MAX_TEL]

  Declare local (static) data here ...
- static char **lookup_fname** [2050]
- static double **Az_src**
- static double **Alt_src**
- static double **Az_nom**
- static double **Alt_nom**
- static double **source_offset**
- static MOMENTS ∗ **pixmom** = NULL
- static struct ebias_cor_data **ebias**
- static int **tel_types_change** = 0
- static int **stat_type** [MAX_TEL_TYPES+2]
- static int **init_hist_for_type** [MAX_TEL_TYPES+2]
- static int **init_hist_global** = 0
- struct basic_ntuple **bnt**

7.62.1 Detailed Description

Code for analysis of simulated (and reconstructed) showers within the framework of the read_hess program.

Users wanting to make use of such analysis should modify the user_∗ functions provided here or the do_user_ana() function. Except for the do_user_ana() function and the user_set_...() functions, all functions are declared as static to emphasize that their interfaces can be changed here to the user's desires.

Author

Konrad Bernloehr

Date

initial version: August 2006

2006 to 2023

7.62.2 Function Documentation

7.62.2.1 ebias_correction()

```
double ebias_correction (
            double lgE )
```

Ask for a correction to log10(reconstructed energy), if available.

Returns

Bias in log10(energy), to be subtracted from log10(energy), or 0.

7.62.2.2 eval_cut_param()

```
static double eval_cut_param (
            double * cut,
            double lgE ) [static]
```

Evaluate energy-dependent cut parameters with.

Parameters

cut[0]	the cut parameter at 1 TeV (lgE=0),
cut[1]	the slope of the cut parameters versus lgE,
cut[2]	the minimum cut parameter,
cut[3]	the maximum cut parameter.

7.62.2.3 expected_max_distance()

```
static double expected_max_distance (
            double E,
            double theta,
            double height ) [static]
```

Expected distance of the shower maximum from the detector for gamma rays, based on simple analytical formula and exponential atmospheric profile.

Parameters

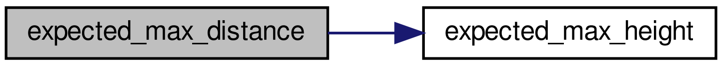
E	The energy of the shower [TeV].
theta	Then zenith angle of the shower [radians].
height	The height above sea level of the experiment [m].

Returns

Distance of shower maximum from detector [m]

References expected_max_height().

Here is the call graph for this function:



7.62.2.4 expected_max_height()

```
static double expected_max_height (
             double E,
             double theta,
             double height )  [static]
```

Expected height of the shower maximum above the detector for gamma rays, based on simple analytical formula and exponential atmospheric profile.

Parameters

E	The energy of the shower [TeV].
theta	Then zenith angle of the shower [radians].
height	The height above sea level of the experiment [m].

Returns

Height of shower maximum above detector [m]

Referenced by expected_max_distance().

7.62.2.5 img_norm()

```
static int img_norm (
             double w,
             double l,
             double A,
             double lgA,
             double rc,
             int tel_type,
             double * scrw,
             double * scrl,
             double * scw,
             double * scl,
             double * sce,
             double * scer,
             double * rco,
             double * rcor,
             double * dimgo,
             double * dimgor )  [static]
```

Get scaled + reduced scaled image parameters (both HEGRA and HESS type scaling) as well as energy scaling from the lookups.

All variables for the results are optional. For variables which are of no interest, pass a NULL pointer.

Parameters

w	Image width [rad].
l	Image length [rad].
A	Image amplitude [peak p.e.].
lgA	log10(A)
rc	Reconstructed core distance.
tel_type	Telescope type (for multiple lookups).
scrw	Variable getting the scaled reduced width (HESS style).
scrl	Variable getting the scaled reduced length (HESS style).
scw	Variable getting the scaled width (HEGRA style).
scl	Variable getting the scaled length (HEGRA style).
sce	Variable getting the expected energy [TeV] for the given amplitude at the given core distance.
scer	Variable getting the relative fluctuation of energy/amplitude at this point.
rco	Variable getting the expected core distance based on width/length and amplitude.
rcor	Variable getting the relative error in the core distance estimate.
dimgo	Variable getting the expected distance in the image (as for rco).
dimgor	Variable getting the relative error in the image distance estimate.

7.62.2.6 init_telescope_types()

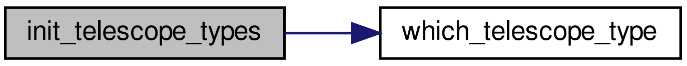
```
static void init_telescope_types (
            AllHessData * hsdata )  [static]
```

Initialize what of type each telescope is.

In normal simulation data this is only needed once but in complex merged (via merge_simtel) data the necessary info may not be available for all of them when types for the first of them is needed.

References simtel_run_header_struct::ntel, simtel_camera_settings_struct::num_mirrors, telescope_type, and which_telescope_type().

Here is the call graph for this function:



7.62.2.7 interp()

```
static void interp (
            double x,
            double * v,
            int n,
            int * ipl,
            double * rpl )  [static]
```

Linear interpolation with binary search algorithm.

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) 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

x	Input: the requested coordinate
v	Input: tabulated coordinates at data points
n	Input: number of data points
ipl	Output: the number of the data point following the requested coordinate in the given sorting (1 <= ipl <= n-1)
rpl	Output: the fraction (x-v[ipl-1])/(v[ipl]-v[ipl-1]) with 0 <= rpl <= 1

Referenced by rpol().

7.62.2.8 rpol()

```
static double rpol (
            double * x,
            double * y,
            int n,
            double xp )  [static]
```

Linear interpolation with binary search algorithm.

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 function calls interp() to find out where to interpolate.

Parameters

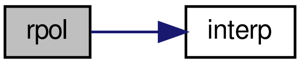
x	Input: Coordinates for data table
y	Input: Corresponding values for data table
n	Input: Number of data points
xp	Input: Coordinate of requested value

Returns

Interpolated value

References interp().

Here is the call graph for this function:



7.62.2.9 user_event_fill()

```
static void user_event_fill (
             AllHessData * hsdata,
             int stage )  [static]
```

Fill (triggered) event specific histograms etc.

< true energy [TeV]

== 0. may happen for calibration events

< Event for desired spectral slope

< true core distance [m]

< reconstructed core distance [m]

< image amplitude [peak p.e.]

< image width [rad]

< image length [rad]

< radius of image c.o.g. in camera plane

< distance of image c.o.g. to source [rad]

< Amplitude and edge distance are ok

References simtel_mc_shower_struct::energy.

7.62.2.10 user_mc_event_fill()

```
static void user_mc_event_fill (
            AllHessData * hsdata )  [static]
```

Work to be done once per shower usage.

Depending on sim_hessarray flags this might be called only for triggered events or also for non-triggered events (default).

References simtel_mc_shower_struct::energy.

7.62.2.11 user_set_flags()

```
void user_set_flags (
            int uf )
```

Set user-defined flags: used to active HESS-style analysis.

Parameters

uf	0: not exactly HESS-style analysis; 1: HESS-style standard cuts; 2: HESS-style hard cuts; 3: HESS-style loose cuts. >=4: HESS-style (no re-scaling) but user-defined cut parameters.

7.62.2.12 user_set_tel_type_param_by_str()

```
int user_set_tel_type_param_by_str (
            const char * str )
```

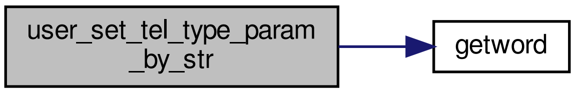
Set telescope type parameters from a string (e.g.

on the command line).

Can be used to set all relevant parameters (others set to 0) or just to switch the active type (no parameters other than the type number).

References getword().

Here is the call graph for this function:



7.62.2.13 user_set_theta_escale()

```
void user_set_theta_escale (
            double * thes )
```

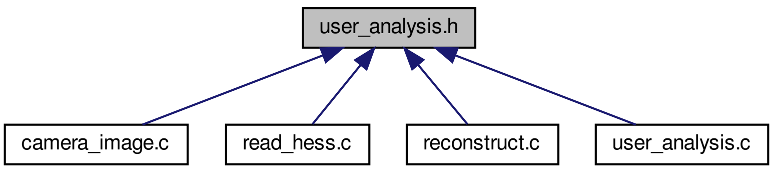
By default the angular acceptance is the 80% containment radius.

Performance may improve by using a smaller radius at low energies (stricter cut) and a larger radius at high energies (looser cut). This sets an additional lg(E) dependent scaling factor.

7.63 user_analysis.h File Reference

Pass data between hessio main program (read_hess) and analysis code.

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



Data Structures

- struct user_parameters

Typedefs

- typedef struct user_parameters **UserParameters**

Functions

- void **user_init_parameters** (void)
- struct user_parameters ∗ **user_get_parameters** (int itype)
- void user_set_lookup_file (const char ∗fname)

  Override the automatic naming for lookup files.
- void user_set_histogram_file (const char ∗fname)

  Override the automatic naming for histogram files.
- void user_set_telescope_type (int itype)

  Select a specific telescope type for setting user parameters.
- int user_set_tel_type_param_by_str (const char ∗str)

  Set telescope type parameters from a string (e.g.
- int which_telescope_type (const CameraSettings ∗cam_set)

  Find out to which telescope type a telescope belongs, by best matching in the required parameters.
- int user_get_type (int itel)

  Get the best matching telescope type for a given telescope index.
- void user_set_spectrum (double di)

  Set the difference between generated MC spectrum and the assumed source spectrum.
- void user_set_impact_range (double ∗impact_range)

  Set the acceptable ranges for reconstructed impact positions.
- void user_set_true_impact_range (double ∗true_impact_range)

  Set the acceptable ranges for true impact positions.
- void user_set_max_core_distance (double rt)

  Set the maximum core distance for telescopes if their images should be used beyond geometrical reconstruction.
- void user_set_min_amp (double a)

  Set the minimum amplitude of images usable for the analysis.
- void user_set_tail_cuts (double tcl, double tch, int lref, double minfrac)

  Set the lower and upper tail cuts for the standard two-level tail-cut scheme.
- void user_set_min_pix (int mpx)

  Set the minimum number of significant pixels in usable images.
- void user_set_reco_flag (int rf)

  Set the reconstruction level flag ('-r' option in read_hess).
- void user_set_tel_img (int tmn, int tmx)

  Set the minimum and maximum number of usable images for events used in analysis.
- void user_set_tel_list (size_t min_tel, size_t ntel, int ∗tel_id)

  You may have alternative selections of (fewer) telescopes.
- void user_set_max_theta (double thmax, double thscale, double thmin)

  Set the maximum angle between source and reconstructed shower direction.
- void user_set_de_cut (double ∗dec)

  The dE cut can be made more or less strict by a scale parameter which should be 1.0 by default and is below 1 for a stricter cut and above 1 for a looser cut.
- void user_set_de2_cut (double ∗de2c)

  Since the dE2 cut is not always of any help with default cut parameters, you can change the parameter to your needs.
- void user_set_hmax_cut (double hmaxc)

  The hmax cut can be made or or less strict by a scale parameter which should be 1.0 by default and is below 1 for a stricter cut and above 1 for a looser cut.
- void user_set_shape_cuts (double wmin, double wmax, double lmin, double lmax)

Set shape cut parameters.
- void user_set_width_max_cut (double ∗wmx)

  Set energy dependent scaled width limit.
- void user_set_length_max_cut (double ∗lmx)

  Set energy dependent scaled length limit.
- void user_set_clipping (double dc)

  Set the maximum radius to be used of a camera.
- void user_set_clipamp (double cpa)

  Set the maximum amplitude in a pixel.
- void **user_set_verbosity** (int v)
- void user_set_flags (int uf)

  Set user-defined flags: used to active HESS-style analysis.
- void **user_set_auto_lookup** (int al)
- void user_set_theta_escale (double ∗the)

  By default the angular acceptance is the 80% containment radius.
- void **user_set_diffuse_mode** (int dm, double oar[ ])
- void **user_set_integrator** (int scheme)
- void **user_set_integ_window** (int nsum, int noff, int ps_opt)
- void **user_set_integ_threshold** (int ithg, int itlg)
- void user_set_trg_req (int trg_req)

  Set the required trigger type(s) as a bit pattern.
- void **user_set_integ_no_rescale** (int no)
- void **user_set_calib_scale** (double s)
- void **user_set_nb_radius** (double ∗r)
- void **user_set_nxt_radius** (double r)
- void **user_set_pixel_stats** (int on)
- void user_set_focal_length (double f)

  Set the telescope effective focal length.
- int **user_selected_event** (void)
- int **do_user_ana** (AllHessData ∗hsdata, unsigned long item_type, int stage)

7.63.1 Detailed Description

Pass data between hessio main program (read_hess) and analysis code.

Author

Konrad Bernloehr

Date

2006 to 2023

7.63.2 Function Documentation

7.63.2.1 user_set_flags()

```
void user_set_flags (
            int uf )
```

Set user-defined flags: used to active HESS-style analysis.

Parameters

uf	0: not exactly HESS-style analysis; 1: HESS-style standard cuts; 2: HESS-style hard cuts; 3: HESS-style loose cuts. >=4: HESS-style (no re-scaling) but user-defined cut parameters.

7.63.2.2 user_set_tel_type_param_by_str()

```
int user_set_tel_type_param_by_str (
            const char * str )
```

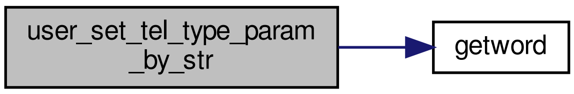
Set telescope type parameters from a string (e.g.

on the command line).

Can be used to set all relevant parameters (others set to 0) or just to switch the active type (no parameters other than the type number).

References getword().

Here is the call graph for this function:



7.63.2.3 user_set_theta_escale()

```
void user_set_theta_escale (
            double * thes )
```

By default the angular acceptance is the 80% containment radius.

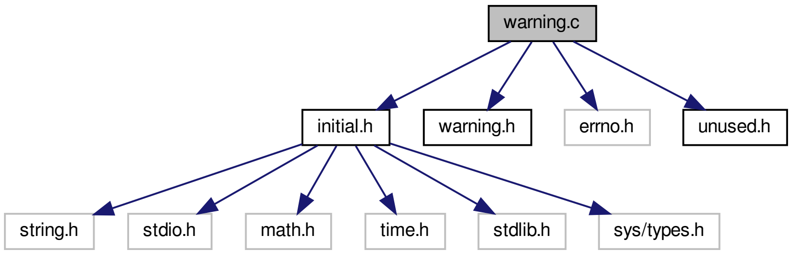
Performance may improve by using a smaller radius at low energies (stricter cut) and a larger radius at high energies (looser cut). This sets an additional lg(E) dependent scaling factor.

7.64 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 "unused.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 warn_f_warning (const char ∗msgtext, const char ∗msgorigin, int msglevel, int msgno)

  Issue a warning to screen or other configured target.
- int set_warning (int level, int mode)

  Set a specific warning level and mode.
- int **set_default_warning** (int level, int mode)
- void warning_status (int ∗plevel, int ∗pmode)

  Inquire status of warning settings.
- void set_logging_function (void(∗user_function)(const char ∗, const char ∗, int, int))

  Set user-defined function for logging warnings and errors.
- void **set_default_logging_function** (void(∗user_function)(const char ∗, const char ∗, int, int))
- int set_log_file (const char ∗fname)

  Set a new log file name and save it in local storage.
- 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 flush_output ()

    Flush buffered output.
- void set_output_function (void(∗user_function)(const char ∗))

    Set a user-defined function as the function to be used for normal text output.
- void **set_default_output_function** (void(∗user_function)(const char ∗))
- void set_aux_warning_function (char ∗(∗auxfunc)(void))

    Set an auxiliary function for warnings.
- void **set_default_aux_warning_function** (char ∗(∗auxfunc)(void))

Variables

- static struct warn_specific_data **warn_defaults**

7.64.1 Detailed Description

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

Author

Konrad Bernloehr

Date

2001 to 2023

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 auxiliary function for time/date etc.
 bit 6: use the auxiliary function output as origin string
        if no explicit origin was supplied.
 bit 7: use syslog().
------------------------------------------------------------
```

7.64.2 Function Documentation

7.64.2.1 flush_output()

```
void flush_output (
             void  )
```

Flush buffered output.

Output is flushed, no matter if it is standard output or a special output function;

Returns

(none)

7.64.2.2 set_aux_warning_function()

```
void set_aux_warning_function (
             char *(*)(void) auxfunc )
```

Set an auxiliary function for warnings.

This function may be used to insert time and date or origin etc. at the beginning of the warning text.

Parameters

auxfunc	– Pointer to a function taking no argument and returning a character string.

Returns

(none)

7.64.2.3 set_log_file()

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

Set a new log file name and save it in local storage.

If there was a log file with a different name opened previously, close it.

Parameters

fname	New name of log file for warnings

Returns

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

7.64.2.4 set_logging_function()

```
void set_logging_function (
            void(*)(const char *, const char *, int, int) user_function )
```

Set user-defined function for logging warnings and errors.

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)

7.64.2.5 set_output_function()

```
void set_output_function (
            void(*)(const char *) user_function )
```

Set a user-defined function as the function to be used for normal text output.

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)

7.64.2.6 set_warning()

```
int set_warning (
            int level,
            int mode )
```

Set a specific warning level and mode.

Parameters

level	Warnings with level below this are ignored.
mode	To screen, to file, with user function ...

Returns

0 if ok, -1 if level and/or mode could not be set.

7.64.2.7 warn_f_output_text()

```
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.

Parameters

text	A text string to be displayed.

Returns

(none)

7.64.2.8 warn_f_warning()

```
void warn_f_warning (
            const char * msgtext,
            const char * msgorigin,
            int msglevel,
            int msgno )
```

Issue a warning to screen or other configured target.

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

msgtext	Warning or error text.
msgorigin	Optional origin (e.g. function name) or NULL.
msglevel	Level of message importance: negative: debugging if needed, 0-9: informative, 10-19: warning, 20-29: error.
msgno	Number of message or 0.

Returns

(none)

7.64.2.9 warning_status()

```
void warning_status (
             int * plevel,
             int * pmode )
```

Inquire status of warning settings.

Parameters

plevel	Pointer to variable for storing current level.
pmode	Pointer to store the current warning mode.

Returns

(none)

7.64.3 Variable Documentation

7.64.3.1 warn_defaults

```
struct warn_specific_data warn_defaults  [static]
```

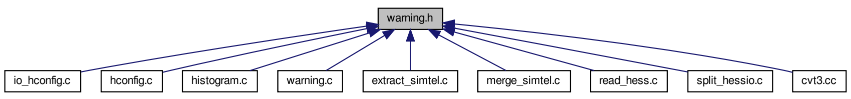
Initial value:

```
=
{
   0,
   1+8,
   "",
   "warning.log",
   "",
   0,
   NULL,
   NULL,
   NULL,
   NULL,
   0
}
```

7.65 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 **WARNING_ORIGIN** (char ∗) NULL
- #define **Information**(string) warn_f_warning(string,WARNING_ORIGIN,0,0)
- #define **Warning**(string) warn_f_warning(string,WARNING_ORIGIN,10,0)
- #define **Error**(string) warn_f_warning(string,WARNING_ORIGIN,20,0)
- #define **Output**(string) warn_f_output_text(string)

Functions

- void warn_f_warning (const char ∗text, const char ∗origin, int level, int msgno)

  Issue a warning to screen or other configured target.
- int set_warning (int level, int mode)

  Set a specific warning level and mode.
- int **set_default_warning** (int level, int mode)
- void warning_status (int ∗plevel, int ∗pmode)

  Inquire status of warning settings.
- void set_logging_function (void(∗user_function)(const char ∗, const char ∗, int, int))

  Set user-defined function for logging warnings and errors.
- void **set_default_logging_function** (void(∗user_function)(const char ∗, const char ∗, int, int))
- int set_log_file (const char ∗fname)

  Set a new log file name and save it in local storage.
- 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 flush_output (void)

  Flush buffered output.
- void set_output_function (void(∗user_function)(const char ∗))

  Set a user-defined function as the function to be used for normal text output.
- void **set_default_output_function** (void(∗user_function)(const char ∗))
- void set_aux_warning_function (char ∗(∗auxfunc)(void))

  Set an auxiliary function for warnings.
- void **set_default_aux_warning_function** (char ∗(∗auxfunc)(void))
- char ∗ **warn_f_get_message_buffer** (void)

7.65.1 Detailed Description

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

Author

Konrad Bernloehr

Date

2001 to 2010

7.65.2 Function Documentation

7.65.2.1 flush_output()

```
void flush_output (
             void  )
```

Flush buffered output.

Output is flushed, no matter if it is standard output or a special output function;

Returns

(none)

7.65.2.2 set_aux_warning_function()

```
void set_aux_warning_function (
             char *(*)(void) auxfunc )
```

Set an auxiliary function for warnings.

This function may be used to insert time and date or origin etc. at the beginning of the warning text.

Parameters

auxfunc	– Pointer to a function taking no argument and returning a character string.

Returns

(none)

7.65.2.3 set_log_file()

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

Set a new log file name and save it in local storage.

If there was a log file with a different name opened previously, close it.

Parameters

fname	New name of log file for warnings

Returns

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

7.65.2.4 set_logging_function()

```
void set_logging_function (
            void(*)(const char *, const char *, int, int) user_function )
```

Set user-defined function for logging warnings and errors.

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)

7.65.2.5 set_output_function()

```
void set_output_function (
            void(*)(const char *) user_function )
```

Set a user-defined function as the function to be used for normal text output.

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)

7.65.2.6 set_warning()

```
int set_warning (
            int level,
            int mode )
```

Set a specific warning level and mode.

Parameters

level	Warnings with level below this are ignored.
mode	To screen, to file, with user function ...

Returns

0 if ok, -1 if level and/or mode could not be set.

7.65.2.7 warn_f_output_text()

```
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.

Parameters

text	A text string to be displayed.

Returns

(none)

7.65.2.8 warn_f_warning()

```
void warn_f_warning (
            const char * msgtext,
            const char * msgorigin,
            int msglevel,
            int msgno )
```

Issue a warning to screen or other configured target.

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

msgtext	Warning or error text.
msgorigin	Optional origin (e.g. function name) or NULL.
msglevel	Level of message importance: negative: debugging if needed, 0-9: informative, 10-19: warning, 20-29: error.
msgno	Number of message or 0.

Returns

(none)

7.65.2.9 warning_status()

```
void warning_status (
            int * plevel,
            int * pmode )
```

Inquire status of warning settings.

Parameters

plevel	Pointer to variable for storing current level.
pmode	Pointer to store the current warning mode.

Returns

(none)

Index