hessio

2014-06-23

Generated by Doxygen 1.8.5

Wed Jun 25 2014 17:35:16

Contents

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

http://www.mpi-hd.mpg.de/hfm/~bernlohr/HESS/Software/hessio/

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

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_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.
- 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.
- 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 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`.
- 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

Namespace Index

3.1 Namespace List

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

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Module Documentation

6.1 The filterio program

Collaboration diagram for The filterio program:

Data Structures

- struct iostats

Macros

- #define **__STDC_LIMIT_MACROS** 1

Functions

- void **syntax** (const char *prg)

6.1.1 Detailed Description

6.2 The listio program

Functions

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

  Main function.

6.2.1 Detailed Description

6.2.2 Function Documentation

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

Main function.

The main function of the listio program.

References allocate_io_buffer(), fileopen(), find_io_block(), _struct_IO_BUFFER::input_file, list_io_blocks(), list_sub_items(), _struct_IO_BUFFER::max_length, read_io_block(), and set_ev_reg_std().

6.3 The statio program

Collaboration diagram for The statio program:

Data Structures

- struct iostats

Macros

- #define **__STDC_LIMIT_MACROS** 1

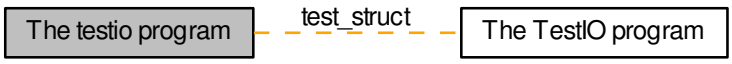
Functions

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

6.3.1 Detailed Description

6.4 The testio program

Collaboration diagram for The testio program:



Data Structures

- struct test_struct

Typedefs

- typedef struct test_struct **TEST_DATA**

Functions

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

  Compare elements of test data structures.
- int write_test1 (TEST_DATA *data, IO_BUFFER *iobuf)

  Write test data with single-element functions.
- int read_test1 (TEST_DATA *data, IO_BUFFER *iobuf)

  Read test data with single-element functions.
- int write_test2 (TEST_DATA *data, IO_BUFFER *iobuf)

  Write test data with vector functions as far as possible.
- int read_test2 (TEST_DATA *data, IO_BUFFER *iobuf)

  Read test data with vector functions as far as possible.
- int write_test3 (TEST_DATA *data, IO_BUFFER *iobuf)

  Write test data in nested items.
- int read_test3 (TEST_DATA *data, IO_BUFFER *iobuf)

  Read test data as a nested tree.
- void syntax (const char *prg)

  Replacement for function missing on OS-9.
- int main (int argc, char **argv)

  Main function for I/O test program.

Variables

- static int **care_long**
- static int **care_int**
- static int **care_short**

6.4.1 Detailed Description

6.4.2 Function Documentation

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

Compare elements of test data structures.

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

Parameters

data1	first data structure
data2	second data structure

Returns

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

Referenced by main().

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

Main function for I/O test program.

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

References allocate_io_buffer(), _struct_IO_BUFFER::byte_order, datacmp(), _struct_IO_BUFFER::extended, fileclose(), fileopen(), find_io_block(), _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::output_file, read_io_block(), read_test1(), read_test2(), read_test3(), syntax(), write_test1(), write_test2(), and write_test3().

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

Read test data with single-element functions.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for get_item_end())

References get_count(), get_count16(), get_count32(), get_int32(), get_item_begin(), get_item_end(), get_long(), get_long_string(), get_real(), get_scount(), get_scount16(), get_scount32(), get_short(), get_string(), get_uint32(), get_var_string(), and _struct_IO_ITEM_HEADER::type.

Referenced by main().

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

Read test data with vector functions as far as possible.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for get_item_end())

References get_count(), get_item_begin(), get_item_end(), get_long(), get_long_string(), get_scount(), get_scount16(), get_scount32(), get_string(), get_var_string(), get_vector_of_byte(), get_vector_of_int(), get_vector_of_int32(), get_vector_of_long(), get_vector_of_real(), get_vector_of_short(), get_vector_of_uint32(), and _struct_IO_ITEM_HEADER::type.

Referenced by main().

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

Read test data as a nested tree.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for get_item_end())

References get_count(), get_item_begin(), get_item_end(), get_long(), get_long_string(), get_scount(), get_scount16(), get_scount32(), get_string(), get_var_string(), get_vector_of_byte(), get_vector_of_int(), get_vector_of_int32(), get_vector_of_long(), get_vector_of_real(), get_vector_of_short(), get_vector_of_uint32(), next_subitem_type(), rewind_item(), search_sub_item(), and _struct_IO_ITEM_HEADER::type.

Referenced by main().

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

Write test data with single-element functions.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (O.K.), <0 (error as for put_item_end())

References _struct_IO_ITEM_HEADER::ident, put_count(), put_count16(), put_count32(), put_int32(), put_item_begin(), put_item_end(), put_long(), put_long_string(), put_real(), put_scount(), put_scount16(), put_scount32(), put_short(), put_string(), put_uint32(), put_var_string(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

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

Write test data with vector functions as far as possible.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for put_item_end())

References _struct_IO_ITEM_HEADER::ident, put_count(), put_count16(), put_count32(), put_item_begin(), put_item_end(), put_long(), put_long_string(), put_scount(), put_scount16(), put_scount32(), put_string(), put_var_string(), put_vector_of_byte(), put_vector_of_int(), put_vector_of_int32(), put_vector_of_long(), put_vector_of_real(), put_vector_of_short(), put_vector_of_uint32(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

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

Write test data in nested items.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

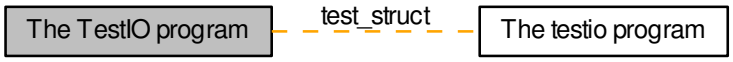
0 (ok), <0 (error as for put_item_end())

References _struct_IO_ITEM_HEADER::ident, put_count(), put_count16(), put_count32(), put_item_begin(), put_item_end(), put_long(), put_long_string(), put_scount(), put_scount16(), put_scount32(), put_string(), put_var_string(), put_vector_of_byte(), put_vector_of_int(), put_vector_of_int32(), put_vector_of_long(), put_vector_of_real(), put_vector_of_short(), put_vector_of_uint32(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

6.5 The TestIO program

Collaboration diagram for The TestIO program:



Data Structures

- struct test_struct

Macros

- #define **__STDC_LIMIT_MACROS** 1

Typedefs

- typedef struct test_struct **TEST_DATA**

Functions

- int datacmp (TEST_DATA &data1, TEST_DATA &data2)

  Compare elements of test data structures.
- int write_test1 (TEST_DATA &data, EventIO &iobuf)

  Write test data with single-element functions.
- int read_test1 (TEST_DATA &data, EventIO &iobuf)

  Read test data with single-element functions.
- int write_test2 (TEST_DATA &data, EventIO &iobuf)

  Write test data with vector functions as far as possible.
- int read_test2 (TEST_DATA &data, EventIO &iobuf)

  Read test data with vector functions as far as possible.
- void **Information** (const char *text)
- void **Warning** (const char *text)
- void **Error** (const char *text)
- int write_test3 (TEST_DATA &data, EventIO &iobuf)

  Write test data in nested items.
- int read_test3 (TEST_DATA &data, EventIO &iobuf)

  Read test data as a nested tree.
- int **write_test_ex** (EventIO &iobuf)
- void **syntax** (const char *prg)
- int main (int argc, char **argv)

  Main function for I/O test program.

Variables

- static int **care_long**
- static int **care_int**
- static int **care_short**

6.5.1 Detailed Description

6.5.2 Function Documentation

6.5.2.1 int datacmp (TEST_DATA & *data1,* TEST_DATA & *data2*)

Compare elements of test data structures.

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

Parameters

data1	first data structure
data2	second data structure

Returns

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

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

Main function for I/O test program.

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

References eventio::EventIO::Buffer(), _struct_IO_BUFFER::byte_order, eventio::EventIO::CloseOutput(), datacmp(), fileopen(), eventio::EventIO::Find(), eventio::EventIO::OpenInput(), eventio::EventIO::OpenOutput(), eventio::EventIO::Read(), read_test1(), read_test2(), read_test3(), syntax(), write_test1(), write_test2(), and write_test3().

6.5.2.3 int read_test1 (TEST_DATA & *data,* EventIO & *iobuf*)

Read test data with single-element functions.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for get_item_end())

References eventio::EventIO::Item::Done(), eventio::EventIO::Item::GetBool(), eventio::EventIO::Item::GetCount16(), eventio::EventIO::Item::GetDouble(), eventio::EventIO::Item::GetInt16(), eventio::EventIO::Item::GetInt32(), eventio::EventIO::Item::GetInt64(), eventio::EventIO::Item::GetReal(), eventio::EventIO::Item::GetSCount16(), eventio::EventIO::Item::GetString(), eventio::EventIO::Item::GetUint16(), eventio::EventIO::Item::GetUint32(), eventio::EventIO::Item::GetUint64(), eventio::EventIO::Item::GetUint8(), and eventio::EventIO::Item::Status().

6.5.2.4 int read_test2 (TEST_DATA & *data*, EventIO & *iobuf*)

Read test data with vector functions as far as possible.

Parameters

data	Test data structure
iobuf	I/O buffer

Returns

0 (ok), <0 (error as for get_item_end())

References eventio::EventIO::Item::Done(), eventio::EventIO::Item::GetBool(), eventio::EventIO::Item::GetDouble(), eventio::EventIO::Item::GetInt16(), eventio::EventIO::Item::GetInt32(), eventio::EventIO::Item::GetInt64(), eventio::EventIO::Item::GetReal(), eventio::EventIO::Item::GetString(), eventio::EventIO::Item::GetUint16(), eventio::EventIO::Item::GetUint32(), eventio::EventIO::Item::GetUint64(), eventio::EventIO::Item::GetUint8(), and eventio::EventIO::Item::Status().

6.5.2.5 int read_test3 (TEST_DATA & *data*, EventIO & *iobuf*)

Read test data as a nested tree.

Parameters

data	Test data structure
iobuf	I/O buffer

Returns

0 (ok), <0 (error as for get_item_end())

References eventio::EventIO::Item::Done(), eventio::EventIO::Item::GetBool(), eventio::EventIO::Item::GetDouble(), eventio::EventIO::Item::GetInt16(), eventio::EventIO::Item::GetInt32(), eventio::EventIO::Item::GetInt64(), eventio::EventIO::Item::GetReal(), eventio::EventIO::Item::GetString(), eventio::EventIO::Item::GetUint16(), eventio::EventIO::Item::GetUint32(), eventio::EventIO::Item::GetUint64(), eventio::EventIO::Item::GetUint8(), eventio::EventIO::Item::NextSubItemType(), eventio::EventIO::Item::Rewind(), eventio::EventIO::Item::Search(), and eventio::EventIO::Item::Status().

6.5.2.6 int write_test1 (TEST_DATA & *data*, EventIO & *iobuf*)

Write test data with single-element functions.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (O.K.), <0 (error as for put_item_end())

References eventio::EventIO::Item::Done(), eventio::EventIO::Item::PutCount16(), eventio::EventIO::Item::PutDouble(), eventio::EventIO::Item::PutInt16(), eventio::EventIO::Item::PutInt32(), eventio::EventIO::Item::PutInt64(), eventio::EventIO::Item::PutReal(), eventio::EventIO::Item::PutSCount16(), eventio::EventIO::Item::PutString(), eventio::EventIO::Item::PutUint16(), eventio::EventIO::Item::PutUint32(), eventio::EventIO::Item::PutUint64(), eventio::EventIO::Item::PutUint8(), and eventio::EventIO::Item::Status().

6.5.2.7 int write_test2 (TEST_DATA & *data*, EventIO & *iobuf*)

Write test data with vector functions as far as possible.

Parameters

data	Pointer to test data structure
iobuf	Pointer to I/O buffer

Returns

0 (ok), <0 (error as for put_item_end())

References eventio::EventIO::Item::Done(), eventio::EventIO::Item::PutDouble(), eventio::EventIO::Item::PutInt16(), eventio::EventIO::Item::PutInt32(), eventio::EventIO::Item::PutInt64(), eventio::EventIO::Item::PutReal(), eventio::EventIO::Item::PutString(), eventio::EventIO::Item::PutUint16(), eventio::EventIO::Item::PutUint32(), eventio::EventIO::Item::PutUint64(), eventio::EventIO::Item::PutUint8(), and eventio::EventIO::Item::Status().

6.5.2.8 int write_test3 (TEST_DATA & *data*, EventIO & *iobuf*)

Write test data in nested items.

Parameters

data	Test data structure
iobuf	I/O buffer

Returns

0 (ok), <0 (error as for put_item_end())

References eventio::EventIO::Item::Done(), eventio::EventIO::Item::PutDouble(), eventio::EventIO::Item::PutInt16(), eventio::EventIO::Item::PutInt32(), eventio::EventIO::Item::PutInt64(), eventio::EventIO::Item::PutReal(), eventio::EventIO::Item::PutString(), eventio::EventIO::Item::PutUint16(), eventio::EventIO::Item::PutUint32(), eventio::EventIO::Item::PutUint64(), eventio::EventIO::Item::PutUint8(), and eventio::EventIO::Item::Status().

6.6 The check_trgmask program

Functions

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

6.6.1 Detailed Description

6.7 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**

6.7.1 Detailed Description

6.7.2 Function Documentation

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

Main program.

Main program function of extract_hess.c program.

References allocate_io_buffer(), copy_item_to_io_block(), fileclose(), fileopen(), find_io_block(), get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::output_file, read_io_block(), reset_io_block(), _struct_IO_ITEM_HEADER::type, and write_io_block().

6.8 The gen_trgmask program

Functions

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

6.8.1 Detailed Description

6.9 The merge_simtel program

Data Structures

- struct map_tel_struct

  Structure with per output telescope information keeping track of prerequisites.

Functions

- 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, 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 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)
- 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 }

6.9.1 Detailed Description

6.9.2 Function Documentation

6.9.2.1 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.

References fileclose(), fileopen(), find_io_block(), _struct_IO_BUFFER::input_file, read_io_block(), read_trgmask(), trgmask_fill_hashed(), and _struct_IO_ITEM_HEADER::type.

Referenced by main().

6.9.3 Variable Documentation

6.9.3.1 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().

6.9.3.2 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(), find_out_tel_idx(), main(), and merge_data_from_io_block().

6.9.3.3 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(), and merge_data_from_io_block().

6.10 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.
- #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**
- typedef struct next_file_struct **NextFile**

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

  Main program.

Variables

- struct basic_ntuple **bnt**
- static int **interrupted**
- static int **dst_processing**
- struct basic_ntuple **bnt**
- static int **interrupted**
- static int **dst_processing**

6.10.1 Detailed Description

6.10.2 Macro Definition Documentation

6.10.2.1 #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.

6.10.2.2 #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.

Referenced by main().

6.10.3 Function Documentation

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

Main program.

Main program function of read_hess.c program.

References allocate_io_buffer(), hess_shower_parameter::Alt, hess_mc_shower_struct::altitude, hess_tel_image_struct::amplitude, angle_between(), hess_shower_parameter::Az, hess_mc_shower_struct::azimuth, book_histogram(), CALIB_SCALE, calibrate_pixel_amplitude(), hess_event_data_struct::central, hess_tel_image_struct::clip_amp, hess_mc_shower_struct::cmax, hess_tel_image_struct::cut_id, Histogram_Extension::ddata, hess_mc_shower_struct::emax, hess_shower_parameter::energy, hess_mc_shower_struct::energy, hess_mc_event_struct::event, histogram::extension, fileclose(), fileopen(), fill_histogram(), fill_histogram_by_ident(), find_io_block(), find_tel_idx(), find_trgmask(), free_histogram(), get_histogram_by_ident(), getword(), H_CHECK_MAX, H_MAX_TEL, hesscam_ps_plot(), hess_mc_shower_struct::hmax, hess_tel_image_struct::hot_amp, hess_tel_image_struct::hot_pixel, _struct_IO_ITEM_HEADER::ident, hess_tel_event_data_struct::image_pixels, hess_tel_event_data_struct::img, _struct_IO_BUFFER::input_file, IO_TYPE_HESS_XTRGMASK, hess_tel_event_adc_struct::known, hess_tel_image_struct::known, hess_tel_image_struct::l, basic_ntuple::lg_e, line_point_distance(), list_ntuple(), user_parameters::lref, hess_tel_event_data_struct::max_image_sets, _struct_IO_BUFFER::max_length, mc_event_fill(), hess_mc_event_struct::mc_pesum, user_parameters::min_amp, user_parameters::min_pix, user_parameters::minfrac, hess_shower_parameter::mscl, hess_shower_parameter::mscw, basic_ntuple::n_img, hess_run_header_struct::ntel, hess_tel_image_struct::num_hot, hess_tel_event_data_struct::num_image_sets, hess_shower_parameter::num_img, hess_camera_settings_struct::num_mirrors, hess_mc_pe_sum_struct::num_pe, hess_tel_monitor_struct::num_ped_slices, hess_event_data_struct::num_tel, hess_central_event_data_struct::num_teltrg, hess_shower_parameter::num_trg, _struct_IO_BUFFER::output_file, hess_tel_image_struct::phi, hess_pixel_list::pixels, hess_tel_image_struct::pixels, hess_tel_event_data_struct::pixtm, hess_mc_shower_struct::primary_id, print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_phot(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixelset(), print_hess_run_stat(), print_hess_runheader(), print_hess_tel_monitor(), print_histograms(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), hess_tel_event_data_struct::raw, RAWDATA_FLAG, read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_phot(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_tel_monitor(), read_hess_trackset(), read_histograms(), read_input_lines(), read_io_block(), read_trgmask(), reconstruct(), reset_io_block(), hess_shower_parameter::result_bits, hess_run_header_struct::run, select_calibration_channel(), set_disabled_pixels(), hess_event_data_struct::shower, skip_io_block(), hess_mc_run_header_struct::spectral_index, stop_-

signal_function(), user_parameters::tailcut_low, hess_run_header_struct::tel_id, hess_camera_settings_struct::tel_id, hess_camera_organisation_struct::tel_id, hess_pixel_setting_struct::tel_id, hess_pixel_disabled_struct::tel_id, hess_camera_software_setting_struct::tel_id, hess_tracking_setup_struct::tel_id, hess_pointing_correction_struct::tel_id, hess_tel_event_adc_struct::tel_id, hess_pixel_timing_struct::tel_id, hess_tel_image_struct::tel_id, hess_tel_event_data_struct::tel_id, hess_tracking_event_data_struct::tel_id, hess_tel_monitor_struct::tel_id, hess_laser_calib_data_struct::tel_id, tel_idx, hess_run_header_struct::tel_pos, hess_event_data_struct::teldata, hess_central_event_data_struct::teltrg_list, hess_central_event_data_struct::teltrg_type_mask, hess_event_data_struct::trackdata, trgmask_entry::trg_mask, user_parameters::trg_req, trgmask_fill_hashed(), _struct_IO_ITEM_HEADER::type, user_get_type(), user_set_clipamp(), user_set_clipping(), user_set_de2_cut(), user_set_de_cut(), user_set_flags(), user_set_histogram_file(), user_set_hmax_cut(), user_set_length_max_cut(), user_set_lookup_file(), user_set_max_theta(), user_set_min_amp(), user_set_min_pix(), user_set_reco_flag(), user_set_shape_cuts(), user_set_spectrum(), user_set_tail_cuts(), user_set_tel_img(), user_set_tel_list(), user_set_tel_type_param_by_str(), user_set_telescope_type(), user_set_theta_escale(), user_set_trg_req(), user_set_width_max_cut(), verbose, hess_tel_image_struct::w, which_telescope_type(), write_dst_histos(), write_hess_event(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_shower(), write_histograms(), write_io_block(), hess_tel_image_struct::x, hess_shower_parameter::xc, hess_mc_event_struct::xcore, hess_shower_parameter::xmax, hess_mc_shower_struct::xmax, hess_tel_image_struct::y, hess_shower_parameter::yc, hess_mc_event_struct::ycore, and hess_tel_event_adc_struct::zero_sup_mode.

6.10.3.2 void stop_signal_function (int *isig*)

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

Parameters

isig	Signal number.

Returns

(none)

6.11 The read_hess_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)
- 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**

6.11.1 Detailed Description

6.11.2 Macro Definition Documentation

6.11.2.1 #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.

Referenced by main().

6.11.3 Function Documentation

6.11.3.1 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.

Referenced by hesscam_ps_plot(), main(), and user_event_fill().

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

Main program.

Main program function of read_hess.c program.

References allocate_io_buffer(), hess_shower_parameter::Alt, hess_mc_shower_struct::altitude, hess_tel_image_struct::amplitude, angle_between(), hess_shower_parameter::Az, hess_mc_shower_struct::azimuth, CALIB_SCALE, calibrate_pixel_amplitude(), hess_mc_shower_struct::cmax, Histogram_Extension::ddata, hess_mc_shower_struct::emax, hess_shower_parameter::energy, hess_mc_shower_struct::energy, hess_mc_event_struct::event, histogram::extension, fileclose(), fileopen(), find_io_block(), find_tel_idx(), H_CHECK_MAX, hesscam_ps_plot(), hess_mc_shower_struct::hmax, hess_tel_image_struct::hot_amp, hess_tel_image_struct::hot_pixel, _struct_IO_ITEM_HEADER::ident, hess_tel_event_data_struct::image_pixels, hess_tel_event_data_struct::img, _struct_IO_BUFFER::input_file, hess_tel_image_struct::l, line_point_distance(), hess_tel_event_data_struct::max_image_sets, _struct_IO_BUFFER::max_length, hess_mc_event_struct::mc_pesum, hess_shower_parameter::mscl, hess_shower_parameter::mscw, hess_run_header_struct::ntel, hess_tel_image_struct::num_hot, hess_tel_event_data_struct::num_image_sets, hess_shower_parameter::num_img, hess_mc_pe_sum_struct::num_pe, hess_tel_monitor_struct::num_ped_slices, hess_event_data_struct::num_tel, hess_shower_parameter::num_trg, hess_tel_image_struct::phi, hess_pixel_list::pixels, hess_tel_image_struct::pixels, hess_tel_event_data_struct::pixtm, hess_mc_shower_struct::primary_id, print_hess_camorgan(), print_hess_camsettings(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixelset(), print_hess_run_stat(), print_hess_runheader(), print_hess_tel_monitor(), print_tel_block(), print_tel_photons(), hess_tel_event_data_struct::raw, read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_phot(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_tel_monitor(), read_hess_trackset(), read_histograms(), read_input_lines(), read_io_block(), user_parameters::reco_flag, reset_io_block(), hess_shower_parameter::result_bits, hess_run_header_struct::run, hess_event_data_struct::shower, skip_io_block(), hess_mc_run_header_struct::spectral_index, stop_signal_function(), hess_run_header_struct::tel_id, hess_camera_settings_struct::tel_id, hess_camera_organisation_struct::tel_id, hess_pixel_setting_struct::tel_id, hess_pixel_disabled_struct::tel_id, hess_camera_software_setting_struct::tel_id, hess_tracking_setup_struct::tel_id, hess_pointing_correction_struct::tel_id, hess_tel_event_adc_struct::tel_id, hess_pixel_timing_struct::tel_id, hess_tel_image_struct::tel_id, hess_tel_event_data_struct::tel_id, hess_tracking_event_data_struct::tel_id, hess_tel_monitor_struct::tel_id, hess_laser_calib_data_struct::tel_id, hess_run_header_struct::tel_pos, hess_event_data_struct::teldata, hess_event_data_struct::trackdata, _struct_IO_ITEM_HEADER::type, verbose, hess_tel_image_struct::w, hess_tel_image_struct::x, hess_shower_parameter::xc, hess_mc_event_struct::xcore, hess_shower_parameter::xmax, hess_mc_shower_struct::xmax, hess_tel_image_struct::y, hess_shower_parameter::yc, and hess_mc_event_struct::ycore.

6.11.3.3 void stop_signal_function (int *isig*)

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

Parameters

isig	Signal number.

Returns

(none)

6.12 The hdata2hbook program (cvt2)

Functions

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

  Main program.

6.12.1 Detailed Description

6.12.2 Function Documentation

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

Main program.

References display_all_histograms(), histogram::entries, get_first_histogram(), histogram::ident, histogram::nbins, histogram::nbins_2d, histogram::next, sort_histograms(), histogram::title, histogram::type, verbose, and write_all_histograms().

6.13 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)

6.13.1 Detailed Description

6.14 The add_histograms program

Functions

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

  Main program.

6.14.1 Detailed Description

6.14.2 Function Documentation

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

Main program.

References display_all_histograms(), histogram::entries, get_first_histogram(), getword(), histogram::ident, histogram::nbins, histogram::nbins_2d, histogram::next, sort_histograms(), syntax(), histogram::title, histogram::type, verbose, and write_all_histograms().

6.15 The fcat program

Macros

- #define **BSIZE** 8192

Functions

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

6.15.1 Detailed Description

6.16 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 }

Functions

- string **particle_type** (SpecType sp)
- double **Crab_Unit** (double E)
- static double **cu** (double x)
- 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)
- 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

6.16.1 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.

6.17 The list_histogram program

Functions

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

  Main program.

6.17.1 Detailed Description

6.17.2 Function Documentation

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

Main program.

References display_all_histograms(), display_histogram(), histogram::entries, get_first_histogram(), get_histogram_by_ident(), histogram::ident, histogram::nbins, histogram::nbins_2d, histogram::next, print_histogram(), sort_histograms(), histogram::title, histogram::type, and verbose.

Chapter 7

Namespace Documentation

7.1 eventio Namespace Reference

The classes of this interface belong to the namespace "eventio".

Data Structures

- class EventIO

  This class provides the embedded buffer, the file I/O interface and the toplevel item access.

Typedefs

- typedef class EventIO::Item **EventIO_Item**

7.1.1 Detailed Description

The classes of this interface belong to the namespace "eventio".

Chapter 8

Data Structure Documentation

8.1 _struct_IO_BUFFER Struct Reference

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

`#include <io_basic.h>`

Data Fields

- unsigned char * buffer
  Pointer to allocated data space.
- long buflen
  Usable length of data space.
- long **r_remaining**
- long w_remaining
  Byte available for reading/writing.
- BYTE * data
  Position for next get.../put...
- int is_allocated
  Indicates if buffer is allocated by eventio.
- int item_level
  Current level of nesting of items.
- long item_length [MAX_IO_ITEM_LEVEL]
  Length of each level of items.
- long sub_item_length [MAX_IO_ITEM_LEVEL]
  Length of its sub-items.
- long item_start_offset [MAX_IO_ITEM_LEVEL]
  Where the item starts in buffer.
- int item_extension [MAX_IO_ITEM_LEVEL]
  Where the extension field was used.
- int input_fileno
  For use of read() function for input.
- int output_fileno
  For use of write() function for output.
- FILE * input_file
  For use of stream I/O for input.
- FILE * output_file
  For use of stream I/O for output.

- int(∗ user_function)(unsigned char ∗, long, int)

  For use of special type of I/O.
- int byte_order

  Set if block is not in internal byte order.
- int data_pending

  Set to 1 when header is read but not the data.
- long min_length

  The initial and minimum length of the buffer.
- long max_length

  The maximum length for extending the buffer.
- int aux_count

  May be used for dedicated buffers.
- int regular

  1 if a regular file, 0 not known, -1 not regular
- int extended

  Set to 1 if you want to use the extension field always.
- int sync_err_count

  Count of synchronization errors.
- int sync_err_max

  Maximum accepted number of synchronisation errors.

8.1.1 Detailed Description

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

8.1.2 Field Documentation

8.1.2.1 unsigned char∗ _struct_IO_BUFFER::buffer

Pointer to allocated data space.

Referenced by allocate_io_buffer(), eventio::EventIO::Append(), append_io_block_as_item(), copy_item_to_io_block(), extend_io_buffer(), find_io_block(), free_io_buffer(), get_item_begin(), get_item_end(), list_sub_items(), next_subitem_type(), put_item_begin_with_flags(), put_item_end(), read_io_block(), remove_item(), reset_io_block(), rewind_item(), search_sub_item(), skip_io_block(), unget_item(), unput_item(), and write_io_block().

8.1.2.2 long _struct_IO_BUFFER::buflen

Usable length of data space.

Referenced by allocate_io_buffer(), copy_item_to_io_block(), extend_io_buffer(), find_io_block(), get_item_begin(), put_item_begin_with_flags(), read_io_block(), remove_item(), reset_io_block(), and unput_item().

8.1.2.3 int _struct_IO_BUFFER::byte_order

Set if block is not in internal byte order.

Referenced by allocate_io_buffer(), copy_item_to_io_block(), find_io_block(), get_int32(), get_item_begin(), get_long(), get_short(), get_uint32(), get_vector_of_int32(), get_vector_of_long(), get_vector_of_uint16(), get_vector_of_uint32(), list_io_blocks(), main(), put_int32(), put_long(), put_short(), put_uint32(), and put_vector_of_uint16().

8.1.2.4 BYTE* _struct_IO_BUFFER::data

Position for next get.../put...

Referenced by allocate_io_buffer(), append_io_block_as_item(), copy_item_to_io_block(), extend_io_buffer(), find_io_block(), get_int32(), get_item_begin(), get_item_end(), get_long(), get_long_string(), get_short(), get_string(), get_uint32(), get_var_string(), get_vector_of_byte(), get_vector_of_float(), get_vector_of_int(), get_vector_of_int32(), get_vector_of_long(), get_vector_of_real(), get_vector_of_short(), get_vector_of_uint16(), get_vector_of_uint32(), list_sub_items(), next_subitem_ident(), next_subitem_length(), next_subitem_type(), put_int32(), put_item_begin_with_flags(), put_item_end(), put_long(), put_short(), put_uint32(), put_vector_of_byte(), put_vector_of_uint16(), remove_item(), reset_io_block(), rewind_item(), search_sub_item(), unget_item(), unput_item(), and write_io_block().

8.1.2.5 int _struct_IO_BUFFER::extended

Set to 1 if you want to use the extension field always.

Referenced by allocate_io_buffer(), main(), and put_item_begin_with_flags().

8.1.2.6 FILE* _struct_IO_BUFFER::input_file

For use of stream I/O for input.

Referenced by allocate_io_buffer(), check_autoload_trgmask(), eventio::EventIO::CloseInput(), find_io_block(), eventio::EventIO::HaveInput(), main(), eventio::EventIO::OpenInput(), read_io_block(), and skip_io_block().

8.1.2.7 int _struct_IO_BUFFER::input_fileno

For use of read() function for input.

Referenced by allocate_io_buffer(), find_io_block(), eventio::EventIO::HaveInput(), read_io_block(), and skip_io_block().

8.1.2.8 int _struct_IO_BUFFER::is_allocated

Indicates if buffer is allocated by eventio.

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

Referenced by allocate_io_buffer(), extend_io_buffer(), and free_io_buffer().

8.1.2.9 int _struct_IO_BUFFER::item_extension[MAX_IO_ITEM_LEVEL]

Where the extension field was used.

Referenced by copy_item_to_io_block(), find_io_block(), get_item_begin(), get_item_end(), list_io_blocks(), list_sub_items(), next_subitem_type(), put_item_begin_with_flags(), put_item_end(), read_io_block(), remove_item(), reset_io_block(), rewind_item(), search_sub_item(), eventio::EventIO::size(), unget_item(), and write_io_block().

8.1.2.10 int _struct_IO_BUFFER::item_level

Current level of nesting of items.

Referenced by allocate_io_buffer(), append_io_block_as_item(), copy_item_to_io_block(), extend_io_buffer(), find_io_block(), get_item_begin(), get_item_end(), list_sub_items(), next_subitem_length(), next_subitem_type(), print_histograms(), put_item_begin_with_flags(), put_item_end(), read_io_block(), remove_item(), reset_io_block(),

rewind_item(), search_sub_item(), eventio::EventIO::Item::size(), eventio::EventIO::size(), skip_io_block(), unget-_item(), unput_item(), and write_io_block().

8.1.2.11 long _struct_IO_BUFFER::item_start_offset[MAX_IO_ITEM_LEVEL]

Where the item starts in buffer.

Referenced by allocate_io_buffer(), copy_item_to_io_block(), get_item_begin(), get_item_end(), list_sub_items(), next_subitem_type(), put_item_begin_with_flags(), put_item_end(), remove_item(), rewind_item(), search_sub_-item(), unget_item(), and unput_item().

8.1.2.12 FILE* _struct_IO_BUFFER::output_file

For use of stream I/O for output.

Referenced by allocate_io_buffer(), eventio::EventIO::CloseOutput(), eventio::EventIO::HaveOutput(), main(), merge_data_from_io_block(), eventio::EventIO::OpenOutput(), write_all_histograms(), write_io_block(), and write-_io_block_to_file().

8.1.2.13 int _struct_IO_BUFFER::output_fileno

For use of write() function for output.

Referenced by allocate_io_buffer(), eventio::EventIO::HaveOutput(), and write_io_block().

8.1.2.14 int _struct_IO_BUFFER::sync_err_count

Count of synchronization errors.

Referenced by allocate_io_buffer(), and find_io_block().

8.1.2.15 int _struct_IO_BUFFER::sync_err_max

Maximum accepted number of synchronisation errors.

Referenced by allocate_io_buffer(), and find_io_block().

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

For use of special type of I/O.

Referenced by allocate_io_buffer(), eventio::EventIO::CloseFunction(), eventio::EventIO::CloseInput(), eventio::-EventIO::CloseOutput(), find_io_block(), eventio::EventIO::HaveInput(), eventio::EventIO::HaveOutput(), eventio::-EventIO::OpenFunction(), read_io_block(), skip_io_block(), and write_io_block().

8.1.2.17 long _struct_IO_BUFFER::w_remaining

Byte available for reading/writing.

Referenced by allocate_io_buffer(), append_io_block_as_item(), copy_item_to_io_block(), extend_io_buffer(), find-_io_block(), get_item_begin(), get_item_end(), put_int32(), put_item_begin_with_flags(), put_item_end(), put_-long(), put_short(), put_uint32(), put_vector_of_byte(), put_vector_of_uint16(), remove_item(), reset_io_block(), rewind_item(), search_sub_item(), unget_item(), unput_item(), and write_io_block().

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

- io_basic.h

8.2 _struct_IO_ITEM_HEADER Struct Reference

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

`#include <io_basic.h>`

Data Fields

- unsigned long type

  The type number telling the type of I/O block.
- unsigned version

  The version number used for the block.
- int can_search

  Set to 1 if I/O block consist of sub-blocks only.
- int level

  Tells how many levels deep we are nested now.
- long ident

  Identity number.
- int user_flag

  One more bit in the header available for user data.
- int use_extension

  Non-zero if the extension header field should be used.
- size_t length

  Length of data field, for information only.

8.2.1 Detailed Description

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

8.2.2 Field Documentation

8.2.2.1 int _struct_IO_ITEM_HEADER::can_search

Set to 1 if I/O block consist of sub-blocks only.

Referenced by eventio::EventIO::EventIO(), find_io_block(), get_item_begin(), eventio::EventIO::Item::IsSearchable(), eventio::EventIO::Item::Item(), list_sub_items(), put_item_begin_with_flags(), and search_sub_item().

8.2.2.2 long _struct_IO_ITEM_HEADER::ident

Identity number.

Referenced by begin_read_tel_array(), begin_write_tel_array(), config_binary_envelope_begin(), config_binary_inquire_numbers(), config_binary_read_numbers(), config_binary_write_index(), eventio::EventIO::EventIO(), find_io_block(), get_item_begin(), eventio::EventIO::Item::Ident(), eventio::EventIO::Item::Item(), eventio::EventIO::ItemIdent(), list_io_blocks(), list_sub_items(), main(), merge_data_from_io_block(), next_subitem_ident(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(),

print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_photo_electrons(), print_trgmask(), put_item_begin_with_flags(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_photo_electrons(), read_tel_array_end(), read_tel_array_head(), read_trgmask(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_trgmask().

8.2.2.3 size_t _struct_IO_ITEM_HEADER::length

Length of data field, for information only.

Referenced by get_item_begin(), put_item_begin_with_flags(), and put_item_end().

8.2.2.4 int _struct_IO_ITEM_HEADER::level

Tells how many levels deep we are nested now.

Referenced by copy_item_to_io_block(), eventio::EventIO::Item::Depth(), eventio::EventIO::EventIO(), get_item_begin(), get_item_end(), eventio::EventIO::Item::Item(), list_sub_items(), put_item_begin_with_flags(), put_item_end(), rewind_item(), search_sub_item(), eventio::EventIO::Item::size(), unget_item(), and unput_item().

8.2.2.5 unsigned long _struct_IO_ITEM_HEADER::type

The type number telling the type of I/O block.

Referenced by begin_read_tel_array(), begin_write_tel_array(), check_autoload_trgmask(), config_binary_envelope_begin(), config_binary_envelope_end(), config_binary_inquire_numbers(), config_binary_read_index(), config_binary_read_numbers(), config_binary_read_text(), config_binary_text_length(), config_binary_write_index(), eventio::EventIO::EventIO(), find_io_block(), get_item_begin(), get_item_end(), eventio::EventIO::Item::Item(), eventio::EventIO::ItemType(), list_io_blocks(), list_sub_items(), main(), merge_data_from_io_block(), next_subitem_ident(), next_subitem_length(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), put_item_begin_with_flags(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pixtime(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_input_lines(), read_io_block(), read_photo_electrons(), read_shower_longitudinal(),

read_tel_array_end(), read_tel_array_head(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), read_test3(), read_trgmask(), remove_item(), eventio::EventIO::Item::Search(), search_sub_item(), skip_io_block(), skip_subitem(), eventio::EventIO::Item::Type(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_trgmask().

8.2.2.6 int _struct_IO_ITEM_HEADER::use_extension

Non-zero if the extension header field should be used.

Referenced by eventio::EventIO::EventIO(), find_io_block(), get_item_begin(), eventio::EventIO::Item::Item(), list_sub_items(), put_item_begin_with_flags(), put_item_end(), remove_item(), and unput_item().

8.2.2.7 int _struct_IO_ITEM_HEADER::user_flag

One more bit in the header available for user data.

Referenced by get_item_begin(), eventio::EventIO::Item::Item(), list_io_blocks(), list_sub_items(), put_item_begin_with_flags(), and eventio::EventIO::Item::UserFlag().

8.2.2.8 unsigned _struct_IO_ITEM_HEADER::version

The version number used for the block.

Referenced by begin_read_tel_array(), begin_write_tel_array(), config_binary_envelope_begin(), config_binary_inquire_numbers(), config_binary_read_index(), config_binary_read_numbers(), config_binary_read_text(), config_binary_text_length(), config_binary_write_index(), eventio::EventIO::EventIO(), find_io_block(), get_item_begin(), eventio::EventIO::Item::Item(), eventio::EventIO::ItemVersion(), list_io_blocks(), list_sub_items(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), put_item_begin_with_flags(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pixtime(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_array_end(), read_tel_array_head(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_trgmask(), eventio::EventIO::Item::Version(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(),

write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_-trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower-_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_-trgmask().

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

- io_basic.h

8.3 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.

8.3.1 Detailed Description

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

8.3.2 Field Documentation

8.3.2.1 size_t basic_ntuple::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.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.2 double basic_ntuple::alt

Reconstructed shower direction (Altitude).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.3 double basic_ntuple::alt_true

True shower direction (Altitude).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.4 double basic_ntuple::az

Reconstructed shower direction (Azimuth).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.5 double basic_ntuple::az_true

True shower direction (Azimuth).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.6 double basic_ntuple::chi2_e

Consistency of individual energy estimates as reduced chi**2 value.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.7 double basic_ntuple::lg_e

Log10 of reconstructed energy.

Referenced by list_ntuple(), main(), and user_event_fill().

8.3.2.8 double basic_ntuple::lg_e_true

log10(true energy of primary).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.9 double basic_ntuple::mdisp

Mean DISP (1.

-width/length) of usable images.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.10 double basic_ntuple::mscrl

Mean scaled reduced length.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.11 double basic_ntuple::mscrw

Mean scaled reduced width.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.12 size_t basic_ntuple::n_fail

Number of failed triggers (telescopes expected to trigger).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.13 size_t basic_ntuple::n_img

Number of used images.

Referenced by list_ntuple(), main(), and user_event_fill().

8.3.2.14 size_t basic_ntuple::n_pix

Total number of used pixels in all used images.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.15 size_t basic_ntuple::n_trg

Number of triggered telescopes.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.16 size_t basic_ntuple::n_tsl0

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

Referenced by list_ntuple(), and user_event_fill().

8.3.2.17 int basic_ntuple::primary

Primary particle ID.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.18 double basic_ntuple::rcm

Mean core distance of telescopes used in reconstruction.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.19 int basic_ntuple::run

Simulation run number.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.20 double basic_ntuple::sig_e

Relative error estimate on E (NOT the r.m.s.
of individual estimates).
Referenced by list_ntuple(), and user_event_fill().

8.3.2.21 double basic_ntuple::sig_mscrl

R.m.s.
of scaled reduced lengths of indvidual images.
Referenced by list_ntuple(), and user_event_fill().

8.3.2.22 double basic_ntuple::sig_mscrw

R.m.s.
of scaled reduced widths of individual images.
Referenced by list_ntuple(), and user_event_fill().

8.3.2.23 double basic_ntuple::sig_theta

R.m.s.
of theta of telescopes pairs (if $>$ 2 tel.).
Referenced by list_ntuple(), and user_event_fill().

8.3.2.24 double basic_ntuple::sig_xmax

R.m.s.
of Xmax from individual telescopes/images.
Referenced by list_ntuple(), and user_event_fill().

8.3.2.25 double basic_ntuple::theta

Angle between source position and rec.
shower direction.
Referenced by list_ntuple(), and user_event_fill().

8.3.2.26 double basic_ntuple::tslope

Core distance corrected mean time slope (deg/ns/100 m).
Referenced by list_ntuple(), and user_event_fill().

8.3.2.27 double basic_ntuple::tsphere

R.m.s.
of trigger times from spherical propagation from shower max.
Referenced by list_ntuple(), and user_event_fill().

8.3.2.28 double basic_ntuple::weight

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

Referenced by list_ntuple(), and user_event_fill().

8.3.2.29 double basic_ntuple::xc

Reconstructed core position at detection level (x coordinate).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.30 double basic_ntuple::xc_true

True core position at detection level (x coordinate).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.31 double basic_ntuple::xfirst_true

Atmospheric depth of first interaction.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.32 double basic_ntuple::xmax

Depth of shower maximum.

Referenced by list_ntuple(), and user_event_fill().

8.3.2.33 double basic_ntuple::xmax_true

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

Referenced by list_ntuple(), and user_event_fill().

8.3.2.34 double basic_ntuple::yc

Reconstructed core position at detection level (y coordinate).

Referenced by list_ntuple(), and user_event_fill().

8.3.2.35 double basic_ntuple::yc_true

True core position at detection level (y coordinate).

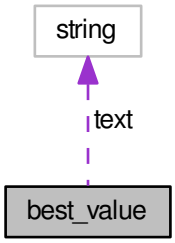
Referenced by list_ntuple(), and user_event_fill().

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

- basic_ntuple.h

8.4 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 vds, double vbr=0., double vgr=0., double var=0., double ver=0.)

Data Fields

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

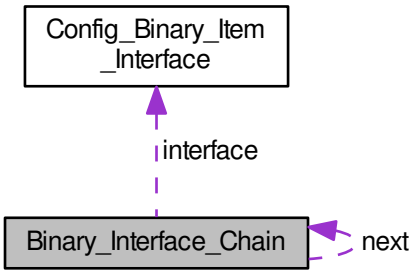
  effective area (for gammas)
- double **lgE**
- double **diff_sens**
- double **bg_rate**
- double **gamma_rate**
- double **angres**
- double **eres**

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

- best_of.cc

8.5 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

8.6 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.

8.6.1 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

8.7 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

8.7.1 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

8.8 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.

8.8.1 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.

8.8.2 Field Documentation

8.8.2.1 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.

Referenced by define_config_binary_interface().

8.8.2.2 int(∗ Config_Binary_Item_Interface::delete_func)(void ∗ptr, int nelem, int item_type)

The function to be called for deleting elements.

Referenced by define_config_binary_interface().

8.8.2.3 int Config_Binary_Item_Interface::elem_size

The size of the elements.

Referenced by define_config_binary_interface(), and init_config().

8.8.2.4 int Config_Binary_Item_Interface::io_item_type

The eventio item type.

Referenced by define_config_binary_interface(), find_config_binary_interface(), and init_config().

8.8.2.5 int(∗ Config_Binary_Item_Interface::list_func)(void ∗bin_item, int item_type)

The optional function for listing element contents.

Referenced by define_config_binary_interface().

8.8.2.6 void∗(∗ Config_Binary_Item_Interface::new_func)(int nelem, int item_type)

The function to be called for allocating elements.

Referenced by define_config_binary_interface(), and init_config().

8.8.2.7 int(∗ Config_Binary_Item_Interface::read_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type)

The function to be called for reading elements from buffer.

Referenced by define_config_binary_interface().

8.8.2.8 int(∗ Config_Binary_Item_Interface::readtext_func)(void ∗bin_item, char ∗text, int item_type)

The function to be called for reading elements from text line.

Referenced by define_config_binary_interface().

8.8.2.9 int(∗ Config_Binary_Item_Interface::write_func)(void ∗bin_item, IO_BUFFER ∗iobuf, int item_type)

The function to be called for writing elements to buffer.

Referenced by define_config_binary_interface().

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

- hconfig.h

8.9 config_specific_data Struct Reference

Data Fields

- char **default_section** [65]

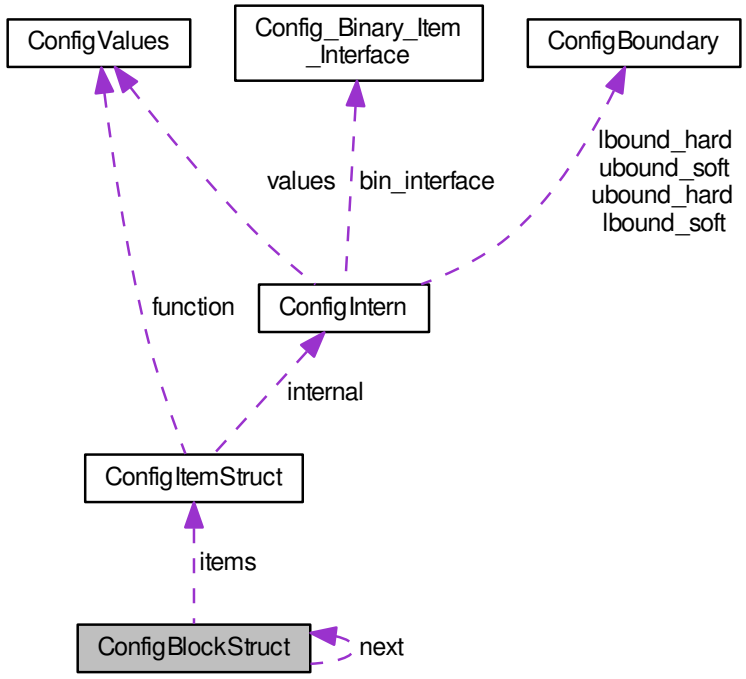
The documentation for this struct was generated from the following file:

- hconfig.c

8.10 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**

8.10.1 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

8.11 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**

8.11.1 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

8.12 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**

8.12.1 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:

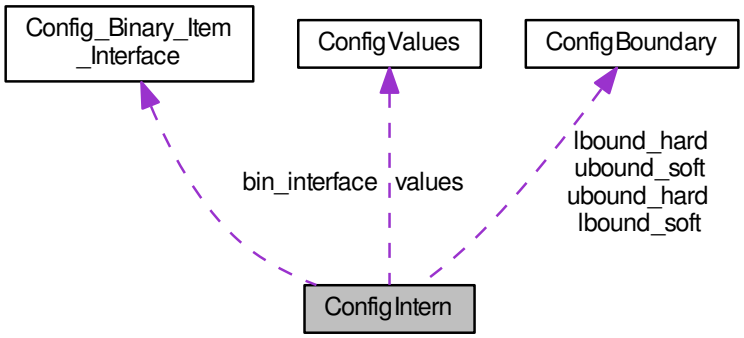
- hconfig.h

8.13 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,.
- 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**

8.13.1 Detailed Description

Configuration elements used only internally.

8.13.2 Field Documentation

8.13.2.1 int ConfigIntern::bound

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

lower hard, or upper hard bound present.

8.13.2.2 int ConfigIntern::elem_size

Size of elements in bytes.

Referenced by init_config().

8.13.2.3 int ConfigIntern::itype

Parameter type code.

Referenced by display_config_item(), do_config(), init_config(), and set_config_values().

8.13.2.4 union ConfigBoundary ConfigIntern::lbound_hard

Used for checking new values.

8.13.2.5 union ConfigBoundary ConfigIntern::lbound_soft

Used for checking new values.

8.13.2.6 int ConfigIntern::locked

Set to 1 if locked.

Referenced by display_config_item(), and reconfig().

8.13.2.7 union ConfigBoundary ConfigIntern::ubound_hard

Used for checking new values.

8.13.2.8 union ConfigBoundary ConfigIntern::ubound_soft

Used for checking new values.

8.13.2.9 struct ConfigValues ConfigIntern::values

Passed to user function.

Referenced by display_config_item(), do_config(), and init_config().

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

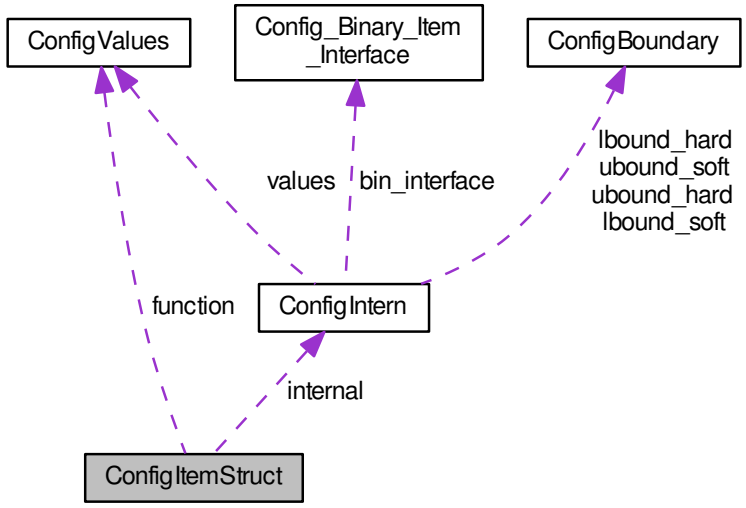
- hconfig.h

8.14 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.

8.14.1 Detailed Description

Configuration as used in definitions of configuration blocks.

8.14.2 Field Documentation

8.14.2.1 void* ConfigItemStruct::data

Data pointer or NULL.

Referenced by display_config_item(), do_config(), init_config(), and set_config_values().

8.14.2.2 int ConfigItemStruct::flags

Additional flag bits.

Referenced by display_config_item(), do_config(), init_config(), and set_config_values().

8.14.2.3 PFIX ConfigItemStruct::function

Associated function or NULL.

Referenced by display_config_item(), and do_config().

8.14.2.4 const char* ConfigItemStruct::initial

Initial values/argument or NULL.

Referenced by display_config_item(), and init_config().

8.14.2.5 struct ConfigIntern ConfigItemStruct::internal

Internal data.

Referenced by display_config_item(), do_config(), init_config(), reconfig(), and set_config_values().

8.14.2.6 const char* ConfigItemStruct::lbound

Lower bound (soft,hard) on values or NULL.

Referenced by display_config_item(), init_config(), and set_config_values().

8.14.2.7 const char* ConfigItemStruct::name

Parameter/function name.

Referenced by display_config_item(), do_config(), f_show_config(), find_config_item(), init_config(), reconfig(), and set_config_values().

8.14.2.8 void* ConfigItemStruct::res1

Placeholder to keep structure size the same.

8.14.2.9 void* ConfigItemStruct::res2

Not used.

8.14.2.10 int ConfigItemStruct::size

Number of elements.

Referenced by display_config_item(), do_config(), init_config(), and set_config_values().

8.14.2.11 const char* ConfigItemStruct::type

Data/function type.

Referenced by display_config_item(), do_config(), and init_config().

8.14.2.12 const char* ConfigItemStruct::ubound

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

Referenced by display_config_item(), init_config(), and set_config_values().

8.14.2.13 PFISS ConfigItemStruct::validate

Function to validate if change is possible or NULL.

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

- hconfig.h

8.15 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.

8.15.1 Detailed Description

Configuration values and supporting data passed to user functions.

8.15.2 Field Documentation

8.15.2.1 int ConfigValues::binary_config

Set to one if binary configuration was used.

8.15.2.2 void* ConfigValues::data_changed

Pointer to the updated values.

Referenced by init_config().

8.15.2.3 void* ConfigValues::data_saved

Pointer to the saved values.

Referenced by do_config(), and init_config().

8.15.2.4 int ConfigValues::elem_size

The size of one element in bytes.

Referenced by display_config_item(), do_config(), and init_config().

8.15.2.5 int ConfigValues::elements

The number of elements it has.

Referenced by init_config().

8.15.2.6 int ConfigValues::itype

Internal item type representation.

Referenced by init_config().

8.15.2.7 int* ConfigValues::list_mod

List of indices to modified elements.

Referenced by do_config(), and init_config().

8.15.2.8 int ConfigValues::max_mod

How many elements can, at most, be modified.

Referenced by do_config(), and init_config().

8.15.2.9 unsigned char* ConfigValues::mod_flag

Vector of size max_mod indicating modified elements.

Referenced by do_config(), and init_config().

8.15.2.10 const char* ConfigValues::name

The name of the element.

Referenced by init_config().

8.15.2.11 int ConfigValues::nmod

How many have been modified.

Referenced by do_config().

8.15.2.12 const char* ConfigValues::section

The section to which it belongs.

Referenced by init_config().

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

- hconfig.h

8.16 ebias_cor_data Struct Reference

Data Fields

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

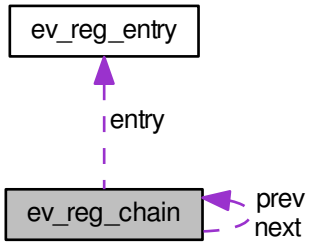
The documentation for this struct was generated from the following file:

- user_analysis.c

8.17 ev_reg_chain Struct Reference

Use a double-linked list for the registry.

Collaboration diagram for ev_reg_chain:



Data Fields

- struct ev_reg_entry * entry

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

8.17.1 Detailed Description

Use a double-linked list for the registry.

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

- eventio_registry.c

8.18 ev_reg_entry Struct Reference

Data Fields

- unsigned long type

  The data block type number.
- char * name

  The data block name (short)
- char * description

  Optional longer description of the data block.

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

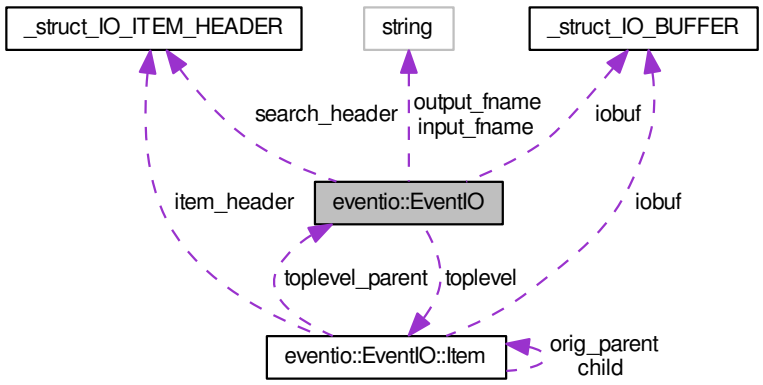
- io_basic.h

8.19 eventio::EventIO Class Reference

This class provides the embedded buffer, the file I/O interface and the toplevel item access.

`#include <EventIO.hh>`

Collaboration diagram for eventio::EventIO:



Data Structures

- class Item

  This (sub-) class provides all the interfaces for putting and getting basic data to and from the embedded buffer.

Public Member Functions

- EventIO (size_t initial_size=65536, size_t max_size=100000000)

  Constructor with initial and maximum sizes for underlying buffer.
- EventIO (IO_BUFFER *bf)

  Constructor that wraps around a pre-existing IO_BUFFER.
- EventIO (const EventIO &eventio)

  The copy constructor is something that might work to some extend, but sooner or later may result in corrupted data, in particular when multi-threaded programs try to write to the same file.
- ~EventIO (void)

  Destructor takes care of finishing any active item, closing input and output files and releasing the underlying I/O buffer (unless the I/O buffer already existed beforehand).
- EventIO & operator= (const EventIO &eventio)

  The same restrictions as for the copy constructor also applies to the assignment operator.
- IO_BUFFER * Buffer (void)

  For more tricky things you can still access the underlying I/O buffer.
- const IO_BUFFER * **Buffer** (void) const
- size_t size (void) const

  Return data size (including 16 or 20 byte header)
- size_t **Length** (void) const
- int OpenInput (const char *fname)

  Open input file.

- int **OpenInput** (const std::string &fname)
- int OpenInput (FILE *f)

  Use externally opened input file or pipe (to be closed externally afterwards).
- int OpenOutput (const char *fname, const char *mode)

  Open Output file.
- int **OpenOutput** (const std::string &fname, const std::string &mode)
- int **OpenOutput** (const char *fname, bool append=false)
- int **OpenOutput** (const std::string &fname, bool append=false)
- int OpenOutput (FILE *f)

  Use externally opened output file or pipe (to be closed externally afterwards).
- int CloseInput (void)

  Close any still open input file for which this object is responsible.
- int CloseOutput (void)

  Close any still open output file or pipe for which this object is responsible.
- int OpenFunction (IO_USER_FUNCTION ufunc)

  Input via user function rather than file.
- int CloseFunction (void)

  Finish with the extern function method for input and/or output.
- bool HaveOutput (void) const

  Check if we have any kind of output assigned.
- bool HaveInput (void) const

  Check if we have any kind of input assigned.
- int Append (const EventIO &ev2)

  Append one full I/O block at the current writing position into another one.
- int Copy (const EventIO::Item &item)

  Copy a sub-item to another I/O buffer as top-level item.
- unsigned long ItemType (void) const

  The item type found by the last Find() call.
- unsigned int ItemVersion (void) const

  The item ident found by the last Find() call.
- long ItemIdent (void) const

  The item ident found by the last Find() call.
- size_t ItemLength (void) const

  The length of the data in the item found by the last Find() call.
- int Find (void)

  Find the next toplevel item in the input stream (file etc.)
- int Read (void)

  Read the next toplevel item from the input stream to the buffer.
- int Skip (void)

  Skip the next toplevel item on the input stream.
- int List (int verbosity=0)

  List the next toplevel item on the input stream.
- int Write (void)

  Explicitly write the current toplevel item to the output stream, if any.
- bool **SetThrow** (bool on=true)
- bool **SetExtended** (bool on=true)

Private Attributes

- IO_BUFFER * **iobuf**
- eventio::EventIO::Item * **toplevel**
- IO_ITEM_HEADER **search_header**
- std::string **input_fname**
- std::string **output_fname**
- bool **local_input**
- bool **local_output**
- bool **throw_on_error**
- bool **external_buffer**

Friends

- class **eventio::EventIO::Item**

8.19.1 Detailed Description

This class provides the embedded buffer, the file I/O interface and the toplevel item access.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 eventio::EventIO::EventIO (size_t *initial_size* = 65536, size_t *max_size* = 100000000)

Constructor with initial and maximum sizes for underlying buffer.

Parameters

initial_size	The initial size of the I/O buffer in bytes.
max_size	The maximum size to which the I/O buffer can be expanded.

References allocate_io_buffer(), _struct_IO_ITEM_HEADER::can_search, _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::level, _struct_IO_BUFFER::max_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, and _struct_IO_ITEM_HEADER::version.

8.19.2.2 eventio::EventIO::EventIO (IO_BUFFER * *bf*)

Constructor that wraps around a pre-existing IO_BUFFER.

It is certainly a bad idea to do this with IO_BUFFERs that are being processed by C (or C-style) code since the chain of parent and top-level items would be incomplete. Thus this constructor should better be applied to IO_BUFFERs at the top-level and the resulting EventIO either destroyed or returned to top-level (e.g. via Done()) before C(-style) code takes over again.

8.19.2.3 eventio::EventIO::EventIO (const EventIO & *eventio*)

The copy constructor is something that might work to some extend, but sooner or later may result in corrupted data, in particular when multi-threaded programs try to write to the same file.

8.19.2.4 eventio::EventIO::~EventIO (void)

Destructor takes care of finishing any active item, closing input and output files and releasing the underlying I/O buffer (unless the I/O buffer already existed beforehand).

References CloseInput(), CloseOutput(), eventio::EventIO::Item::Done(), and free_io_buffer().

8.19.3 Member Function Documentation

8.19.3.1 int eventio::EventIO::CloseFunction (void)

Finish with the extern function method for input and/or output.

Returns

0 (OK), -1 (error)

References _struct_IO_BUFFER::user_function.

Referenced by CloseInput(), and CloseOutput().

8.19.3.2 int eventio::EventIO::CloseInput (void)

Close any still open input file for which this object is responsible.

Returns

0 (OK), -1 (error on closing file or pipe)

References CloseFunction(), fileclose(), _struct_IO_BUFFER::input_file, and _struct_IO_BUFFER::user_function.

Referenced by main(), OpenFunction(), OpenInput(), and ~EventIO().

8.19.3.3 int eventio::EventIO::CloseOutput (void)

Close any still open output file or pipe for which this object is responsible.

Returns

0 (OK), -1 (error on closing file or pipe)

References CloseFunction(), fileclose(), _struct_IO_BUFFER::output_file, and _struct_IO_BUFFER::user_function.

Referenced by main(), OpenFunction(), OpenOutput(), and ~EventIO().

8.19.3.4 bool eventio::EventIO::HaveInput (void) const

Check if we have any kind of input assigned.

Returns

true if any file or pipe or function is assigned for input.

References _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, and _struct_IO_BUFFER::user_function.

8.19.3.5 bool eventio::EventIO::HaveOutput (void) const

Check if we have any kind of output assigned.

Returns

true if any file or pipe or function is assigned for output.

References _struct_IO_BUFFER::output_file, _struct_IO_BUFFER::output_fileno, and _struct_IO_BUFFER::user_function.

8.19.3.6 int eventio::EventIO::OpenFunction (IO_USER_FUNCTION *ufunc*)

Input via user function rather than file.

Use the external function method for input and/or output.

Parameters

ufunc	A user function providing the necessary functionality.

Returns

0 (always)

References CloseInput(), CloseOutput(), and _struct_IO_BUFFER::user_function.

8.19.3.7 int eventio::EventIO::OpenInput (const char * *fname*)

Open input file.

Open input file locally (this object takes responsibility) for closing it afterwards.

Parameters

fname	The name of the input file or "-" for standard input.

Returns

0 (OK), -1 (opening the input file failed)

References CloseInput(), fileopen(), and _struct_IO_BUFFER::input_file.

Referenced by main().

8.19.3.8 int eventio::EventIO::OpenInput (FILE * *f*)

Use externally opened input file or pipe (to be closed externally afterwards).

Parameters

f	A FILE pointer for the input file.

Returns

0 (OK), -1 (NULL file pointer passed)

References CloseInput(), and _struct_IO_BUFFER::input_file.

8.19.3.9 int eventio::EventIO::OpenOutput (const char * *fname*, const char * *mode*)

Open Output file.

Open output file locally (this object takes responsibility) for closing it afterwards.

Parameters

fname	The name of the file to be used for output or "-" for standard output.
mode	The output mode like "w" or "a" (see 'man fopen').

Returns

0 (OK), -1 (opening the output file failed)

References CloseOutput(), fileopen(), and _struct_IO_BUFFER::output_file.

Referenced by main().

8.19.3.10 int eventio::EventIO::OpenOutput (FILE * *f*)

Use externally opened output file or pipe (to be closed externally afterwards).

Parameters

f	A FILE pointer for the output file.

Returns

0 (OK), -1 (NULL file pointer passed)

References CloseOutput(), and _struct_IO_BUFFER::output_file.

8.19.3.11 EventIO & eventio::EventIO::operator= (const EventIO & *eventio*)

The same restrictions as for the copy constructor also applies to the assignment operator.

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

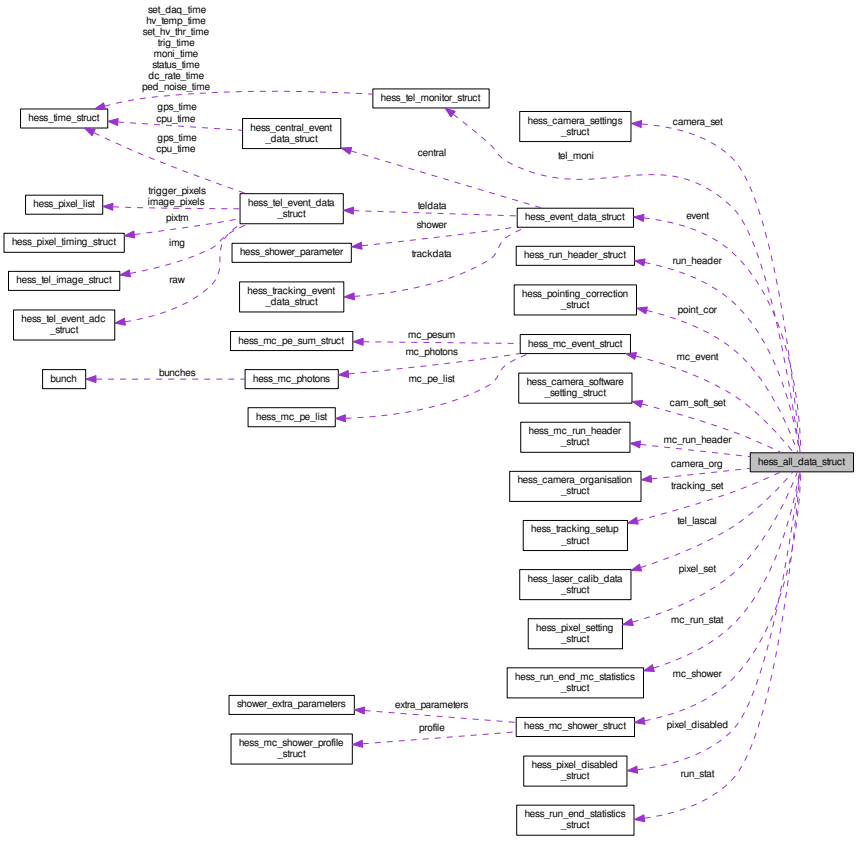
- EventIO.hh
- EventIO.cc

8.20 hess_all_data_struct Struct Reference

Container for all H.E.S.S.

`#include <io_hess.h>`

Collaboration diagram for hess_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]
- RunStat **run_stat**
- MCRunStat **mc_run_stat**

8.20.1 Detailed Description

Container for all H.E.S.S.

data

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

- io_hess.h

8.21 hess_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) in camera.
- int num_gains
  Number of gains per PM.
- int num_sectors
  Number of sectors (trigger groups).
- int drawer [H_MAX_PIX]
  Drawer 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
- 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 ?].

8.21.1 Detailed Description

Logical organisation of camera electronics channels.

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

- io_hess.h

8.22 hess_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 area [H_MAX_PIX]

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

  Pixel diameter (flat-to-flat, [m]).
- double cam_rot

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

  Focal length of optics [m].
- int num_mirrors

  Number of mirror tiles.
- double mirror_area

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

8.22.1 Detailed Description

Definition of camera optics settings.

8.22.2 Field Documentation

8.22.2.1 double hess_camera_settings_struct::mirror_area

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

Referenced by read_hess_camsettings(), user_init(), which_telescope_type(), and write_hess_camsettings().

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

- io_hess.h

8.23 hess_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]

8.23.1 Detailed Description

Software settings used in camera process.

8.23.2 Field Documentation

8.23.2.1 int hess_camera_software_setting_struct::zero_sup_mode

The desired zero suppression mode.

The mode actually used may depend on the data.

Referenced by read_hess_camsoftset(), and write_hess_camsoftset().

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

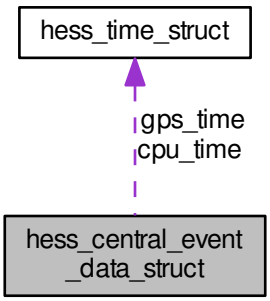
- io_hess.h

8.24 hess_central_event_data_struct Struct Reference

Central trigger event data.

`#include <io_hess.h>`

Collaboration diagram for hess_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][3]

  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 with data.

8.24.1 Detailed Description

Central trigger event data.

8.24.2 Field Documentation

8.24.2.1 int hess_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.)

Referenced by calibrate_amplitude(), merge_data_from_io_block(), read_hess_centralevent(), read_hess_event(), and write_hess_centralevent().

8.24.2.2 int hess_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.)

Referenced by calibrate_amplitude(), merge_data_from_io_block(), read_hess_centralevent(), read_hess_event(), and write_hess_centralevent().

8.24.2.3 float hess_central_event_data_struct::teltrg_time[H_MAX_TEL]

Relative time of trigger signal.

after correction for nominal delay [ns].

Referenced by merge_data_from_io_block(), read_hess_centralevent(), read_hess_event(), write_hess_centralevent(), and write_hess_event().

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

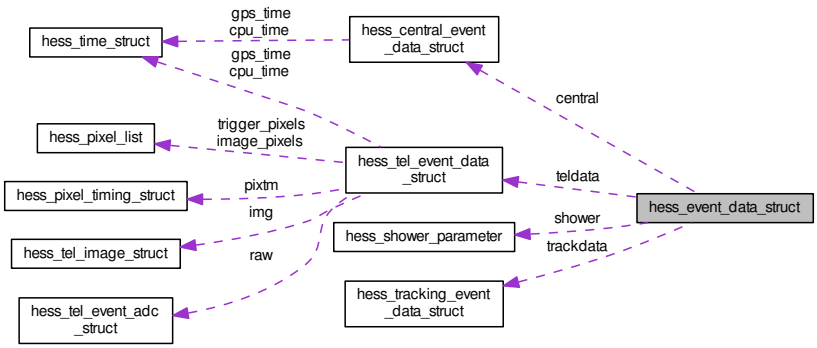
- io_hess.h

8.25 hess_event_data_struct Struct Reference

All data for one event.

`#include <io_hess.h>`

Collaboration diagram for hess_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.

8.25.1 Detailed Description

All data for one event.

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

- io_hess.h

8.26 hess_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]

8.26.1 Detailed Description

Laser calibration data.

8.26.2 Field Documentation

8.26.2.1 double hess_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.

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), read_hess_laser_calib(), and write_hess_laser_-calib().

8.26.2.2 double hess_laser_calib_data_struct::max_int_frac[H_MAX_GAINS]

Maximum fraction of the signal which can be in the fixed integration window.

Referenced by read_hess_laser_calib(), and write_hess_laser_calib().

8.26.2.3 double hess_laser_calib_data_struct::max_pixtm_frac[H_MAX_GAINS]

Maximum fraction of the signal which can be in the pixel timing integration.

Referenced by read_hess_laser_calib(), and write_hess_laser_calib().

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

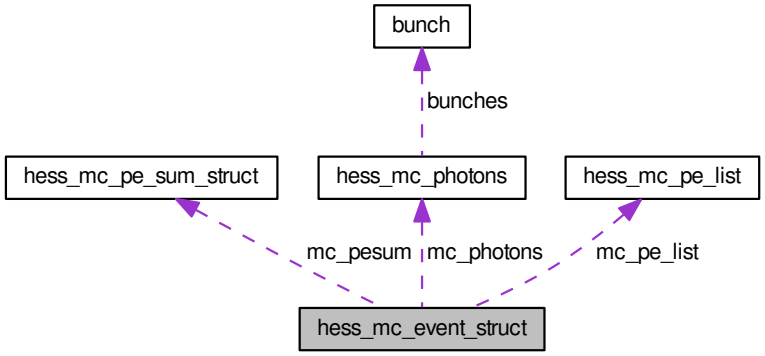
- io_hess.h

8.27 hess_mc_event_struct Struct Reference

Monte Carlo event-specific data.

`#include <io_hess.h>`

Collaboration diagram for hess_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.

- struct hess_mc_pe_sum_struct mc_pesum

  Numbers of / sums of photo-electrons.

- struct hess_mc_photons mc_photons [H_MAX_TEL]

  Raw simulated photons.

- struct hess_mc_pe_list mc_pe_list [H_MAX_TEL]

  List of detected photo-electrons.

8.27.1 Detailed Description

Monte Carlo event-specific data.

8.27.2 Field Documentation

8.27.2.1 double hess_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 merge_data_from_io_block(), read_hess_mc_event(), and write_hess_mc_event().

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

- io_hess.h

8.28 hess_mc_pe_list Struct Reference

Photo-electrons from Monte Carlo 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).

8.28.1 Detailed Description

Photo-electrons from Monte Carlo individually.

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

- io_hess.h

8.29 hess_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.

8.29.1 Detailed Description

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

8.29.2 Field Documentation

8.29.2.1 double hess_mc_pe_sum_struct::photons_atm_qe[H_MAX_TEL]

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

Referenced by merge_data_from_io_block(), read_hess_mc_event(), read_hess_mc_pe_sum(), and write_hess_mc_pe_sum().

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

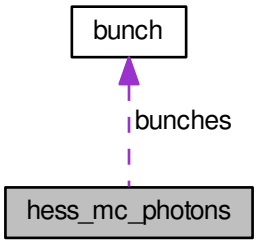
- io_hess.h

8.30 hess_mc_photons Struct Reference

Photons from Monte Carlo.

`#include <io_hess.h>`

Collaboration diagram for hess_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.

8.30.1 Detailed Description

Photons from Monte Carlo.

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

- io_hess.h

8.31 hess_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

  Height of particle injection [m].
- double fixed_int_depth

  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**

8.31.1 Detailed Description

MC run header.

8.31.2 Field Documentation

8.31.2.1 int hess_mc_run_header_struct::shower_prog_id

Recorded data:

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

Referenced by read_hess_mcrunheader(), and write_hess_mcrunheader().

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

- io_hess.h

8.32 hess_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).

8.32.1 Detailed Description

Monte Carlo shower profile (sort of histogram).

8.32.2 Field Documentation

8.32.2.1 int hess_mc_shower_profile_struct::id

Type of profile (also determines units below).

Temptative definitions:

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

The value of k specifies the binning:

- k = 0: The profile is in terms of atmospheric depth along the shower axis.
- k = 1: in terms of vertical atmospheric depth.
- k = 2: in terms of altitude [m] above sea level.

Referenced by read_hess_mc_shower(), and write_hess_mc_shower().

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

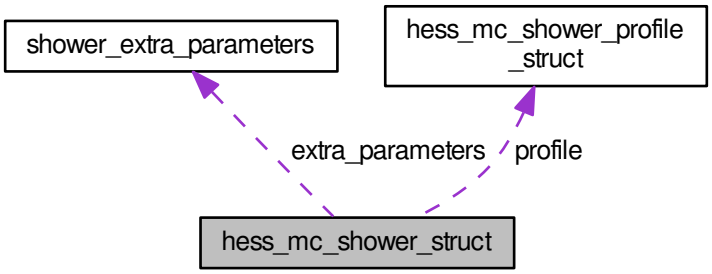
- io_hess.h

8.33 hess_mc_shower_struct Struct Reference

Shower specific data.

`#include <io_hess.h>`

Collaboration diagram for hess_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**

8.33.1 Detailed Description

Shower specific data.

8.33.2 Field Documentation

8.33.2.1 int hess_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.

Referenced by hesscam_ps_plot(), main(), merge_data_from_io_block(), read_hess_mc_shower(), user_event_fill(), user_finish(), user_init(), and write_hess_mc_shower().

8.33.2.2 double hess_mc_shower_struct::xmax

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

Referenced by main(), read_hess_mc_shower(), second_moments(), user_event_fill(), and write_hess_mc_shower().

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

- io_hess.h

8.34 hess_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**
- int **trigger_disabled** [H_MAX_PIX]
- int **num_HV_disabled**
- int **HV_disabled** [H_MAX_PIX]

8.34.1 Detailed Description

Pixels disabled in HV and/or trigger.

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

- io_hess.h

8.35 hess_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.

8.35.1 Detailed Description

Lists of pixels (triggered, selected, etc.)

8.35.2 Field Documentation

8.35.2.1 int hess_pixel_list::code

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

Referenced by merge_data_from_io_block(), read_hess_pixel_list(), and write_hess_pixel_list().

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

- io_hess.h

8.36 hess_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**
- int **trigger_mode**
- 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 in the camera.
- int threshold_DAC [H_MAX_DRAWERS]

  Threshold DAC values set.
- int **ADC_start** [H_MAX_DRAWERS]
- int **ADC_count** [H_MAX_DRAWERS]
- double time_slice

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

Standard integration over so many time slices.

- 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].

8.36.1 Detailed Description

Settings of pixel HV and thresholds.

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

- io_hess.h

8.37 hess_pixel_timing_struct Struct Reference

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.

8.37.1 Field Documentation

8.37.1.1 float hess_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.

Referenced by merge_data_from_io_block(), read_hess_pixtime(), and write_hess_pixtime().

8.37.1.2 int hess_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).

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), merge_data_from_io_block(), read_hess_pixtime(), and write_hess_pixtime().

8.37.1.3 int hess_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.

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), merge_data_from_io_block(), read_hess_pixtime(), and write_hess_pixtime().

8.37.1.4 int hess_pixel_timing_struct::threshold

Minimum base-to-peak raw amplitude difference applied in pixel selection.

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), merge_data_from_io_block(), read_hess_pixtime(), and write_hess_pixtime().

8.37.1.5 float hess_pixel_timing_struct::time_level[H_MAX_PIX_TIMES]

The width and startpos types apply.

above some fraction from base to peak.

Referenced by merge_data_from_io_block(), pixel_timing_analysis(), read_hess_pixtime(), and write_hess_pixtime().

8.37.1.6 float hess_pixel_timing_struct::timval[H_MAX_PIX][H_MAX_PIX_TIMES]

Only the first 'pixels'.

elements are actually filled and stored. Others are undefined.

Referenced by build_list_for_hess_pixtime(), calibrate_amplitude(), calibrate_pixel_amplitude(), merge_data_from_io_block(), pixel_timing_analysis(), read_hess_pixtime(), write_hess_pixtime(), and write_hess_televent().

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

- io_hess.h

8.38 hess_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]

8.38.1 Detailed Description

Pointing correction parameters.

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

- io_hess.h

8.39 hess_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.

8.39.1 Detailed Description

MC end-of-run statistics.

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

- io_hess.h

8.40 hess_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.

8.40.1 Detailed Description

End-of-run statistics.

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

- io_hess.h

8.41 hess_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**

8.41.1 Detailed Description

Run header common to measured and simulated data.

8.41.2 Field Documentation

8.41.2.1 double hess_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.

Referenced by merge_data_from_io_block(), read_hess_runheader(), and write_hess_runheader().

8.41.2.2 double hess_run_header_struct::conv_ref_pos[2]

Reference position for convergent pointing.

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

Referenced by merge_data_from_io_block(), read_hess_runheader(), and write_hess_runheader().

8.41.2.3 double hess_run_header_struct::direction[2]

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

Referenced by mc_event_fill(), merge_data_from_io_block(), read_hess_runheader(), shower_reconstruct(), user_init(), and write_hess_runheader().

8.41.2.4 double hess_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).

Referenced by merge_data_from_io_block(), read_hess_runheader(), and write_hess_runheader().

8.41.2.5 int hess_run_header_struct::reverse_flag

Normal or reverse tracking: 0: Normal, 1: reverse.

Referenced by merge_data_from_io_block(), read_hess_runheader(), and write_hess_runheader().

8.41.2.6 int hess_run_header_struct::run

Recorded data:

Run number.

Referenced by hesscam_ps_plot(), main(), merge_data_from_io_block(), read_hess_runheader(), user_event_fill(), and write_hess_runheader().

8.41.2.7 int hess_run_header_struct::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.

Referenced by merge_data_from_io_block(), read_hess_runheader(), and write_hess_runheader().

8.41.2.8 double hess_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.

Referenced by hesscam_ps_plot(), main(), merge_data_from_io_block(), read_hess_runheader(), second_moments(), shower_reconstruct(), user_event_fill(), user_init(), and write_hess_runheader().

8.41.2.9 int hess_run_header_struct::tracking_mode

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

/Dec. 2000

Referenced by mc_event_fill(), merge_data_from_io_block(), read_hess_runheader(), and write_hess_runheader().

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

- io_hess.h

8.42 hess_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**

8.42.1 Detailed Description

Reconstructed shower parameters.

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

- io_hess.h

8.43 hess_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.

8.43.1 Detailed Description

ADC data (either sampled or sum mode)

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

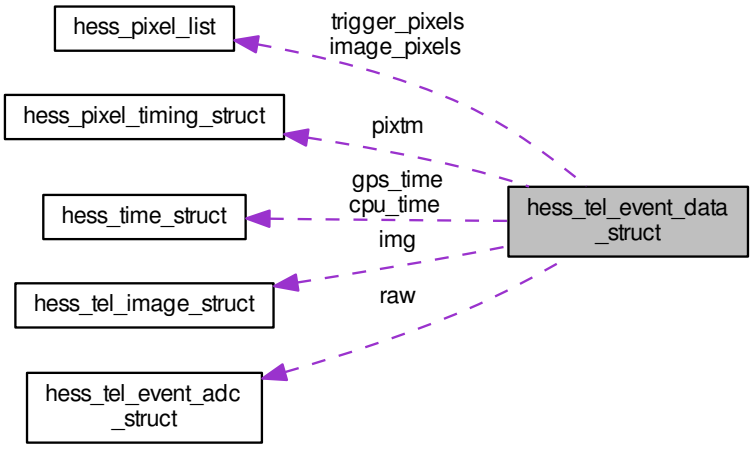
- io_hess.h

8.44 hess_tel_event_data_struct Struct Reference

Event raw and image data from one telescope.

`#include <io_hess.h>`

Collaboration diagram for hess_tel_event_data_struct:



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 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 (1 ... 255, normally: 1).
- 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.
- 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.

8.44.1 Detailed Description

Event raw and image data from one telescope.

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

- io_hess.h

8.45 hess_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.].

8.45.1 Detailed Description

Image parameters.

8.45.2 Field Documentation

8.45.2.1 double hess_tel_image_struct::l

Shape.

Length (major axis) [rad]

Referenced by hesscam_ps_plot(), main(), read_hess_telimage(), second_moments(), shower_reconstruct(), user_event_fill(), and write_hess_telimage().

8.45.2.2 int hess_tel_image_struct::num_hot

Individual pixels.

Number of hottest pixels individually saved

Referenced by main(), read_hess_telimage(), user_event_fill(), and write_hess_telimage().

8.45.2.3 double hess_tel_image_struct::phi

Orientation.

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

Referenced by hesscam_ps_plot(), main(), pixel_timing_analysis(), read_hess_telimage(), second_moments(), shower_reconstruct(), user_event_fill(), and write_hess_telimage().

8.45.2.4 double hess_tel_image_struct::tm_slope

Timing.

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

Referenced by pixel_timing_analysis(), read_hess_telimage(), user_event_fill(), and write_hess_telimage().

8.45.2.5 double hess_tel_image_struct::x

Position.

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

Referenced by hesscam_ps_plot(), main(), pixel_timing_analysis(), read_hess_telimage(), second_moments(), shower_reconstruct(), user_event_fill(), and write_hess_telimage().

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

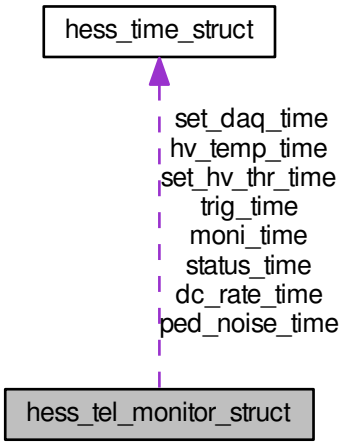
- io_hess.h

8.46 hess_tel_monitor_struct Struct Reference

Monitoring data.

`#include <io_hess.h>`

Collaboration diagram for hess_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**
- 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.
- 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 noise [H_MAX_GAINS][H_MAX_PIX]

  Average noise on ADC sums.
- 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**

8.46.1 Detailed Description

Monitoring data.

8.46.2 Field Documentation

8.46.2.1 long hess_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).

Referenced by read_hess_tel_monitor(), and write_hess_tel_monitor().

8.46.2.2 uint16_t hess_tel_monitor_struct::current[H_MAX_PIX]

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

ADC values of DC current.

Referenced by read_hess_tel_monitor(), and write_hess_tel_monitor().

8.46.2.3 short hess_tel_monitor_struct::drawer_temp[H_MAX_DRAWERS][H_MAX_D_TEMP]

That is left in its raw order:

ADC values.

Referenced by read_hess_tel_monitor(), and write_hess_tel_monitor().

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

- io_hess.h

8.47 hess_time_struct Struct Reference

Breakdown of time into seconds since 1970.0 and nanoseconds.

`#include <io_hess.h>`

Data Fields

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

8.47.1 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

8.48 hess_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.

8.48.1 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

8.49 hess_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**

8.49.1 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.

8.49.2 Field Documentation

8.49.2.1 double hess_tracking_setup_struct::range_low_az

Note: The values may be outside the [0...2∗pi[range.

Referenced by read_hess_trackset(), and write_hess_trackset().

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

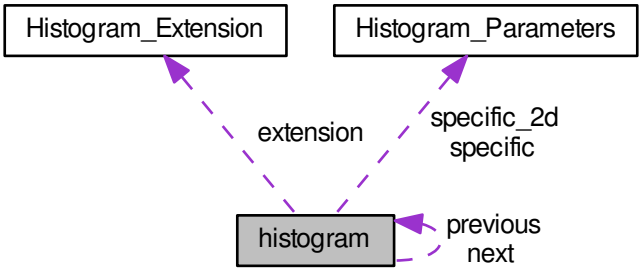
- io_hess.h

8.50 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.

8.50.1 Detailed Description

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

8.50.2 Field Documentation

8.50.2.1 unsigned long histogram::entries

No.

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

Referenced by clear_histogram(), display_2d_histogram(), display_histogram(), fast_stat_histogram(), fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_int_histogram(), fill_real_histogram(), fill_weighted_histogram(), histogram_to_lookup(), histogram_to_root(), list_histograms(), main(), print_histogram(), read_histograms_x(), stat_histogram(), write_dst_histos(), and write_histograms().

8.50.2.2 struct histogram* histogram::next

linked list of histograms.

Referenced by convert_histograms_to_root(), display_all_histograms(), free_all_histograms(), get_histogram_by_ident(), histogram_hashing(), initialize_histogram(), list_histograms(), main(), set_first_histogram(), sort_histograms(), unlink_histogram(), and write_histograms().

8.50.2.3 unsigned long histogram::overflow

No.

of entries above range

Referenced by add_histogram(), clear_histogram(), display_2d_histogram(), display_histogram(), fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_int_histogram(), fill_real_histogram(), fill_weighted_histogram(), histogram_to_lookup(), histogram_to_root(), locate_histogram_fraction(), print_histogram(), read_histograms_x(), and write_histograms().

8.50.2.4 unsigned long histogram::overflow_2d

Same in 2nd coord of 2-D histo.

Referenced by add_histogram(), clear_histogram(), display_2d_histogram(), fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), histogram_to_root(), read_histograms_x(), and write_histograms().

8.50.2.5 unsigned long histogram::tentries

No.

of entries, without """

Referenced by clear_histogram(), display_histogram(), fast_stat_histogram(), fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_int_histogram(), fill_real_histogram(), fill_weighted_histogram(), histogram_to_lookup(), list_histograms(), lookup_int(), lookup_real(), print_histogram(), read_histograms_x(), stat_histogram(), and write_histograms().

8.50.2.6 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 add_histogram(), aux_alloc_histogram(), clear_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_lookup(), histogram_to_root(), list_histograms(), locate_histogram_fraction(), lookup_int(), lookup_real(), main(), print_histogram(), read_histograms_x(), set_ebias_correction(), stat_histogram(), and write_histograms().

8.50.2.7 unsigned long histogram::underflow

No.

of entries below range

Referenced by add_histogram(), clear_histogram(), display_2d_histogram(), display_histogram(), fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_int_histogram(), fill_real_histogram(), fill_weighted_histogram(), histogram_to_lookup(), histogram_to_root(), locate_histogram_fraction(), print_histogram(), read_histograms_x(), and write_histograms().

8.50.2.8 unsigned long histogram::underflow_2d

Same in 2nd coord of 2-D histo.

Referenced by add_histogram(), clear_histogram(), display_2d_histogram(), fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), histogram_to_root(), read_histograms_x(), and write_histograms().

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

- histogram.h

8.51 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.

8.51.1 Detailed Description

A histogram extension only allocated for weighted histograms.

8.51.2 Field Documentation

8.51.2.1 double* Histogram_Extension::ddata

in one of two precisions.

Referenced by add_histogram(), aux_alloc_histogram(), clear_histogram(), display_2d_histogram(), fill_gaps(), fill_weighted_histogram(), free_histo_contents(), gen_image_lookups(), histogram_to_root(), img_norm(), main(), print_histogram(), read_histograms_x(), set_ebias_correction(), stat_histogram(), and user_init().

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

- histogram.h

8.52 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.

8.52.1 Detailed Description

Parameters defining the usable range of coordinates.

8.52.2 Field Documentation

8.52.2.1 struct { ... } Histogram_Parameters::integer

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

Needed for integer-type limits.

Referenced by add_histogram(), alloc_2d_int_histogram(), alloc_int_histogram(), clear_histogram(), display_2d_histogram(), display_histogram(), fast_stat_histogram(), fill_2d_int_histogram(), fill_int_histogram(), histogram_matching(), histogram_to_root(), locate_histogram_fraction(), lookup_int(), print_histogram(), read_histograms_x(), stat_histogram(), and write_histograms().

8.52.2.2 double Histogram_Parameters::inverse_binwidth

1.

/(width_of_one_bin)

Referenced by allocate_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_real_histogram(), fill_weighted_histogram(), and lookup_real().

8.52.2.3 struct { ... } Histogram_Parameters::real

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

Needed for real-type limits.

Referenced by add_histogram(), allocate_histogram(), clear_histogram(), display_2d_histogram(), display_histogram(), fast_stat_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_gaps(), fill_real_histogram(), fill_weighted_histogram(), gen_image_lookups(), histogram_matching(), histogram_to_root(), img_norm(), locate_histogram_fraction(), lookup_real(), print_histogram(), read_histograms_x(), set_ebias_correction(), stat_histogram(), and write_histograms().

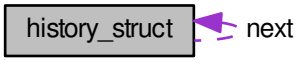
The documentation for this union was generated from the following file:

- histogram.h

8.53 history_struct Struct Reference

Use to build a linked list of configuration history.

Collaboration diagram for history_struct:



Data Fields

- char * **text**
- time_t time

Configuration test.

- struct history_struct * next

  Time when the configuration was entered.

8.53.1 Detailed Description

Use to build a linked list of configuration history.

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

- io_history.c

8.54 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**

8.54.1 Detailed Description

Statistics element for histogram analysis.

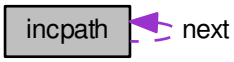
The documentation for this struct was generated from the following file:

- histogram.h

8.55 incpath Struct Reference

An element in a linked list of include paths.

Collaboration diagram for incpath:



Data Fields

- char * path

  The path name.

- struct incpath * next

  The next element.

8.55.1 Detailed Description

An element in a linked list of include paths.

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

- fileopen.c

8.56 iostats Struct Reference

Public Member Functions

- **iostats** (size_t c, uint64_t b, unsigned v)
- **iostats** (size_t c, uint64_t b, unsigned v)

Data Fields

- size_t **count**
- uint64_t **bytes**
- unsigned **version_low**
- unsigned **version_high**

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

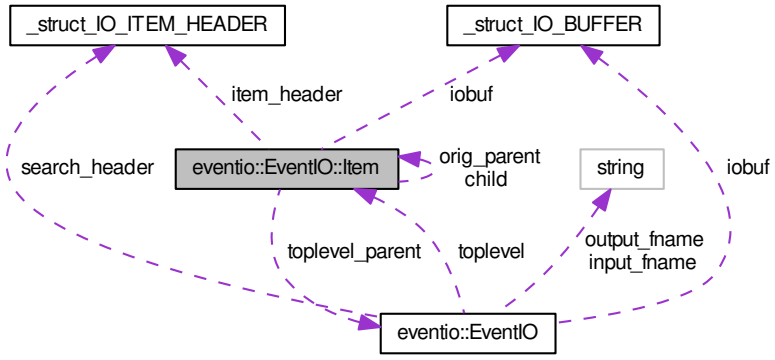
- filterio.cc
- statio.cc

8.57 eventio::EventIO::Item Class Reference

This (sub-) class provides all the interfaces for putting and getting basic data to and from the embedded buffer.

`#include <EventIO.hh>`

Collaboration diagram for eventio::EventIO::Item:



Public Member Functions

- Item (EventIO &ev, const char *method, size_t type=0, size_t version=0, long ident=0, bool user_flag=false, bool extended_flag=false)

  Item constuctor for toplevel item takes the EventIO buffer as first argument.
- Item (Item &parent, const char *method, size_t type=0, size_t version=0, long ident=0, bool user_flag=false, bool extended_flag=false)

  Item constructor for sub-items takes the parent item as first argument.
- Item (const Item &item)

  The item copy constructor is something that might work to some extend, but sooner or later may result in corrupted data, in particular when multi-threaded programs try to write to the same file.
- ~Item (void)

  Item destructor takes care that all sub-items are finished first.
- Item & operator= (const Item &item)

  Restrictions to the copy constructor also apply to the assignment operator.
- int Done (void)

  All put/get operations for this item and all sub-items are done.
- int Status (void) const

  Returns <0 in case of error, 0 for active item, 1 for finished item.
- IO_BUFFER * Buffer (void)

  For more tricky things you can still access the underlying I/O buffer.
- size_t Type (void) const

  Return the item type.
- size_t Version (void) const

  Return the version number.
- long Ident (void) const

  Return the item ID.
- int Depth (void) const

  Return the nesting level depth.
- size_t size (void) const

  Return the data size (excluding header)
- size_t Length (void) const

  Return the length of the data area in the current item.

- bool IsSearchable (void) const

  Return whether an item can be searched for sub-items.
- int Search (size_t sub)

  Search for a specific sub-item.
- int Rewind (void)

  Rewind to beginning of data area.
- int Unget (void)

  Completely undo getting this item, i.e.
- int Unput (void)

  Completely undo putting this item, as if never started.
- int Skip (void)

  Skip a sub-item starting at the current position.
- bool UserFlag (void) const

  Is the user flag set?
- bool **IsExtended** (void) const
- int List (int maxlevel=0, int verbosity=0)

  List item type, length, ID, ... + sub-items.
- const char * TypeName (void)

  Registered name for this type of I/O block.
- const char * Description (void)

  Registered description for this type of I/O block.
- int NextSubItemType (void) const

  Access to information of next sub-item at current reading position.
- size_t NextSubItemLength (void) const

  Access to length information of next sub-item at current reading position.
- long NextSubItemIdent (void) const

  Access to ID information of next sub-item at current reading position.
- uint8_t GetUint8 (void)

  Get operations for bytes (only unsigned flavour implemented).
- void **GetUint8** (uint8_t &v)
- void **GetUint8** (uint8_t *vec, size_t num)
- void GetUint8 (std::vector< uint8_t > &vec, size_t num)

  Get operations for bytes (only unsigned flavour implemented).
- void GetUint8 (std::vector< uint8_t > &vec)

  Get operations for bytes (only unsigned flavour implemented).
- void GetUint8 (std::valarray< uint8_t > &vec, size_t num)

  Get operations for bytes (only unsigned flavour implemented).
- void GetUint8 (std::valarray< uint8_t > &vec)

  Get operations for bytes (only unsigned flavour implemented).
- bool GetBool (void)

  Individual 'bool' elements are stored as unsigned chars holding up to 8 bools.
- void **GetBool** (bool &v)
- void GetBool (std::vector< bool > &vec, size_t num)

  Get operations for bools.
- void GetBool (std::vector< bool > &vec)

  Get operations for bools.
- void GetBool (std::valarray< bool > &vec, size_t num)

  Get operations for bools.
- void GetBool (std::valarray< bool > &vec)

  Get operations for bools.
- uint16_t GetCount16 (void)

Get 'count' type data (unsigned integer stored with variable lenghth).

- uint32_t **GetCount32** (void)
- uintmax_t **GetCount** (void)
- void GetCount (uint16_t *vec, size_t num)

  Get operations for counts (16 bit variant)
- void GetCount (size_t *vec, size_t num)

  Get operations for counts.
- void **GetCount** (uint8_t &v)
- void **GetCount** (uint16_t &v)
- void **GetCount** (size_t &v)
- void GetCount (std::vector< uint16_t > &vec, size_t num)

  Get operations for counts.
- void GetCount (std::vector< uint16_t > &vec)

  Get operations for counts.
- void GetCount (std::vector< size_t > &vec, size_t num)

  Get operations for counts.
- void GetCount (std::vector< size_t > &vec)

  Get operations for counts.
- void GetCount (std::valarray< uint16_t > &vec, size_t num)

  Get operations for counts.
- void GetCount (std::valarray< uint16_t > &vec)

  Get operations for counts.
- void GetCount (std::valarray< size_t > &vec, size_t num)

  Get operations for counts.
- void GetCount (std::valarray< size_t > &vec)

  Get operations for counts.
- int16_t GetSCount16 (void)

  Get 'scount' type data (signed integer stored with variable lenghth).
- int32_t **GetSCount32** (void)
- intmax_t **GetSCount** (void)
- void GetSCount (int16_t *vec, size_t num)

  Get operations for counts.
- void **GetDiffSCount** (int16_t *vec, size_t num)
- void GetSCount (ssize_t *vec, size_t num)

  Get operations for signed counts.
- void **GetDiffSCount** (ssize_t *vec, size_t num)
- void **GetSCount** (int8_t &v)
- void **GetDiffSCount** (int8_t &v)
- void **GetSCount** (int16_t &v)
- void **GetDiffSCount** (int16_t &v)
- void **GetSCount** (ssize_t &v)
- void **GetDiffSCount** (ssize_t &v)
- void GetSCount (std::vector< int16_t > &vec, size_t num)

  Get operations for signed counts.
- void **GetDiffSCount** (std::vector< int16_t > &vec, size_t num)
- void GetSCount (std::vector< int16_t > &vec)

  Get operations for signed counts.
- void **GetDiffSCount** (std::vector< int16_t > &vec)
- void GetSCount (std::vector< ssize_t > &vec, size_t num)

  Get operations for signed counts.
- void **GetDiffSCount** (std::vector< ssize_t > &vec, size_t num)
- void GetSCount (std::vector< ssize_t > &vec)

Get operations for signed counts.

- void **GetDiffSCount** (std::vector< ssize_t > &vec)
- void GetSCount (std::valarray< int16_t > &vec, size_t num)

  Get operations for signed counts.
- void **GetDiffSCount** (std::valarray< int16_t > &vec, size_t num)
- void GetSCount (std::valarray< int16_t > &vec)

  Get operations for signed counts.
- void **GetDiffSCount** (std::valarray< int16_t > &vec)
- void GetSCount (std::valarray< ssize_t > &vec, size_t num)

  Get operations for signed counts.
- void **GetDiffSCount** (std::valarray< ssize_t > &vec, size_t num)
- void GetSCount (std::valarray< ssize_t > &vec)

  Get operations for signed counts.
- void **GetDiffSCount** (std::valarray< ssize_t > &vec)
- int16_t GetInt16 (void)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void **GetInt16** (int16_t &v)
- void **GetInt16** (int &v)
- void **GetInt16** (int16_t *vec, size_t num)
- void **GetInt16** (int *vec, size_t num)
- void GetInt16 (std::vector< int16_t > &vec, size_t num)

  Get operations for signed counts.
- void GetInt16 (std::vector< int > &vec, size_t num)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void GetInt16 (std::vector< int16_t > &vec)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void GetInt16 (std::vector< int > &vec)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void GetInt16 (std::valarray< int16_t > &vec, size_t num)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void GetInt16 (std::valarray< int > &vec, size_t num)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void GetInt16 (std::valarray< int16_t > &vec)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void GetInt16 (std::valarray< int > &vec)

  Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- uint16_t GetUint16 (void)

  Get operations for 'Uint16' item type (unsigned 16-bit integers)
- void **GetUint16** (uint16_t &v)
- void **GetUint16** (size_t &v)
- void **GetUint16** (uint16_t *vec, size_t num)
- void GetUint16 (std::vector< uint16_t > &vec, size_t num)

  Get operations for 'Uint16' item type (unsigned 16-bit integers)
- void **GetUint16** (std::vector< unsigned int > &vec, size_t num)
- void GetUint16 (std::vector< uint16_t > &vec)

  Get operations for 'Uint16' item type (unsigned 16-bit integers)
- void **GetUint16** (std::vector< unsigned int > &vec)
- void GetUint16 (std::valarray< uint16_t > &vec, size_t num)

  Get operations for 'Uint16' item type (unsigned 16-bit integers)
- void **GetUint16** (std::valarray< unsigned int > &vec, size_t num)
- void GetUint16 (std::valarray< uint16_t > &vec)

  Get operations for 'Uint16' item type (unsigned 16-bit integers)

- void **GetUint16** (std::valarray< unsigned int > &vec)
- int32_t GetInt32 (void)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void **GetInt32** (int32_t &v)
- void **GetInt32** (size_t &v)
- void **GetInt32** (long &v)
- void **GetInt32** (int32_t *vec, size_t num)
- void **GetInt32** (long *vec, size_t num)
- void GetInt32 (std::vector< int32_t > &vec, size_t num)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::vector< long > &vec, size_t num)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::vector< int32_t > &vec)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::vector< long > &vec)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::valarray< int32_t > &vec, size_t num)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::valarray< long > &vec, size_t num)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::valarray< int32_t > &vec)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void GetInt32 (std::valarray< long > &vec)

  Get operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- uint32_t GetUint32 (void)

  Get operations for 'Uint32' item type (unsigned 32-bit integers)
- void **GetUint32** (uint32_t &v)
- void **GetUint32** (unsigned long &v)
- void **GetUint32** (uint32_t *vec, size_t num)
- void GetUint32 (std::vector< uint32_t > &vec, size_t num)

  Get operations for 'Uint32' item type (unsigned 32-bit integers)
- void **GetUint32** (std::vector< unsigned long > &vec, size_t num)
- void GetUint32 (std::vector< uint32_t > &vec)

  Get operations for 'Uint32' item type (unsigned 32-bit integers)
- void **GetUint32** (std::vector< unsigned long > &vec)
- void GetUint32 (std::valarray< uint32_t > &vec, size_t num)

  Get operations for 'Uint32' item type (unsigned 32-bit integers)
- void **GetUint32** (std::valarray< unsigned long > &vec, size_t num)
- void GetUint32 (std::valarray< uint32_t > &vec)

  Get operations for 'Uint32' item type (unsigned 32-bit integers)
- void **GetUint32** (std::valarray< unsigned long > &vec)
- int64_t GetInt64 (void)

  Get operations for 'Int64' item type (signed 64-bit integers)
- void **GetInt64** (int64_t &v)
- void **GetInt64** (int64_t *vec, size_t num)
- void GetInt64 (std::vector< int64_t > &vec, size_t num)

  Get operations for 'Int64' item type (signed 64-bit integers)
- void GetInt64 (std::vector< int64_t > &vec)

  Get operations for 'Int64' item type (signed 64-bit integers)
- void GetInt64 (std::valarray< int64_t > &vec, size_t num)

  Get operations for 'Int64' item type (signed 64-bit integers)
- void GetInt64 (std::valarray< int64_t > &vec)

Get operations for 'Int64' item type (signed 64-bit integers)

- uint64_t GetUint64 (void)

  Get operations for 'Uint64' item type (unsigned 64-bit integers)
- void **GetUint64** (uint64_t &v)
- void **GetUint64** (uint64_t *vec, size_t num)
- void GetUint64 (std::vector< uint64_t > &vec, size_t num)

  Get operations for 'Uint64' item type (unsigned 64-bit integers)
- void GetUint64 (std::vector< uint64_t > &vec)

  Get operations for 'Uint64' item type (unsigned 64-bit integers)
- void GetUint64 (std::valarray< uint64_t > &vec, size_t num)

  Get operations for 'Uint64' item type (unsigned 64-bit integers)
- void GetUint64 (std::valarray< uint64_t > &vec)

  Get operations for 'Uint64' item type (unsigned 64-bit integers)
- double GetReal (void)

  Get operations for 'Real' item type (32-bit IEEE float)
- void **GetReal** (float &v)
- void **GetReal** (double &v)
- void **GetReal** (float *vec, size_t num)
- void **GetReal** (double *vec, size_t num)
- void GetReal (std::vector< float > &vec, size_t num)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::vector< double > &vec, size_t num)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::vector< float > &vec)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::vector< double > &vec)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::valarray< float > &vec, size_t num)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::valarray< double > &vec, size_t num)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::valarray< float > &vec)

  Get operations for 'Real' item type (32-bit IEEE float)
- void GetReal (std::valarray< double > &vec)

  Get operations for 'Real' item type (32-bit IEEE float)
- double GetDouble (void)

  Get operations for 'Double' item type (64-bit IEEE double)
- void **GetDouble** (double &v)
- void **GetDouble** (double *vec, size_t num)
- void GetDouble (std::vector< double > &vec, size_t num)

  Get operations for 'Double' item type (64-bit IEEE double)
- void GetDouble (std::vector< double > &vec)

  Get operations for 'Double' item type (64-bit IEEE double)
- void GetDouble (std::valarray< double > &vec, size_t num)

  Get operations for 'Double' item type (64-bit IEEE double)
- void GetDouble (std::valarray< double > &vec)

  Get operations for 'Double' item type (64-bit IEEE double)
- double GetSfloat (void)

  Get operations for 'Sfloat' item type (16-bit OpenGL float)
- void **GetSfloat** (float &v)
- void **GetSfloat** (double &v)

- void **GetSfloat** (double *vec, size_t num)
- void **GetSfloat** (float *vec, size_t num)
- void GetSfloat (std::vector< float > &vec, size_t num)

  Get operations for 'Sfloat' item type (16-bit OpenGL float)
- void **GetSfloat** (std::vector< double > &vec, size_t num)
- void GetSfloat (std::vector< float > &vec)

  Get operations for 'Sfloat' item type (16-bit OpenGL float)
- void **GetSfloat** (std::vector< double > &vec)
- void GetSfloat (std::valarray< float > &vec, size_t num)

  Get operations for 'Sfloat' item type (16-bit OpenGL float)
- void **GetSfloat** (std::valarray< double > &vec, size_t num)
- void GetSfloat (std::valarray< float > &vec)

  Get operations for 'Sfloat' item type (16-bit OpenGL float)
- void **GetSfloat** (std::valarray< double > &vec)
- int GetString (char *s, size_t nmax)

  Get operations for strings with length encoded as Int16.
- int GetString16 (std::string &s)

  Get operation for old-style strings with 16-bit length prefix.
- std::string **GetString16** ()
- int GetString32 (std::string &s)

  Get operation for old-style strings with 32-bit length prefix.
- std::string **GetString32** ()
- int GetString (std::string &s)

  Get operation for strings of any length.
- std::string **GetString** ()
- void GetString (std::vector< std::string > &vec, size_t num)

  Get operation for vectors of strings of any length.
- void GetString (std::vector< std::string > &vec)

  Get operation for vectors of strings of any length.
- void GetString (std::valarray< std::string > &vec, size_t num)

  Get operation for valarrays of strings of any length.
- void GetString (std::valarray< std::string > &vec)

  Get operation for valarrays of strings of any length.
- void PutUint8 (uint8_t b)

  Put operations for bytes (only unsigned flavour implemented).
- void **PutUint8** (const uint8_t *vec, size_t num)
- void PutUint8 (const std::vector< uint8_t > &vec, size_t num)

  Put operations for bytes (only unsigned flavour implemented).
- void **PutUint8** (const std::vector< uint8_t > &vec)
- void PutUint8 (const std::valarray< uint8_t > &vec, size_t num)

  Put operations for bytes (only unsigned flavour implemented).
- void **PutUint8** (const std::valarray< uint8_t > &vec)
- void **PutBool** (bool flag)
- void PutBool (const std::vector< bool > &vec, size_t num)

  Put operations for bools.
- void **PutBool** (const std::vector< bool > &vec)
- void PutBool (const std::valarray< bool > &vec, size_t num)

  Put operations for bools.
- void **PutBool** (const std::valarray< bool > &vec)
- void PutCount16 (size_t n)

  Put operation for 'Count' item type (unsigned integer of variable length);.
- void **PutCount** (const uint16_t *vec, size_t num)

- void **PutCount32** (uint32_t n)
- void **PutCount** (const size_t *vec, size_t num)
- void **PutCount** (size_t n)
- void PutCount (const std::vector< uint16_t > &vec, size_t num)

  Put operations for counts.
- void **PutCount** (const std::vector< uint16_t > &vec)
- void **PutCount** (const std::vector< size_t > &vec, size_t num)
- void **PutCount** (const std::vector< size_t > &vec)
- void PutCount (const std::valarray< uint16_t > &vec, size_t num)

  Put operations for counts.
- void **PutCount** (const std::valarray< uint16_t > &vec)
- void **PutCount** (const std::valarray< size_t > &vec, size_t num)
- void **PutCount** (const std::valarray< size_t > &vec)
- void PutSCount16 (int16_t n)

  Put operation for 'SCount' item type (signed integer of variable length);.
- void **PutSCount** (const int16_t *vec, size_t num)
- void **PutSCount32** (int32_t n)
- void **PutSCount** (const ssize_t *vec, size_t num)
- void **PutSCount** (ssize_t n)
- void PutSCount (const std::vector< int16_t > &vec, size_t num)

  Put operations for counts.
- void **PutDiffSCount** (const std::vector< int16_t > &vec, size_t num)
- void **PutSCount** (const std::vector< int16_t > &vec)
- void **PutDiffSCount** (const std::vector< int16_t > &vec)
- void **PutSCount** (const std::vector< ssize_t > &vec, size_t num)
- void **PutDiffSCount** (const std::vector< ssize_t > &vec, size_t num)
- void **PutSCount** (const std::vector< ssize_t > &vec)
- void **PutDiffSCount** (const std::vector< ssize_t > &vec)
- void PutSCount (const std::valarray< int16_t > &vec, size_t num)

  Put operations for scounts.
- void **PutDiffSCount** (const std::valarray< int16_t > &vec, size_t num)
- void **PutSCount** (const std::valarray< int16_t > &vec)
- void **PutDiffSCount** (const std::valarray< int16_t > &vec)
- void **PutSCount** (const std::valarray< ssize_t > &vec, size_t num)
- void **PutDiffSCount** (const std::valarray< ssize_t > &vec, size_t num)
- void **PutSCount** (const std::valarray< ssize_t > &vec)
- void **PutDiffSCount** (const std::valarray< ssize_t > &vec)
- void PutInt16 (int16_t s)

  Put operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void **PutInt16** (int s)
- void **PutInt16** (const int16_t *vec, size_t num)
- void **PutInt16** (const int *vec, size_t num)
- void PutInt16 (const std::vector< int16_t > &vec, size_t num)

  Put operations for counts.
- void **PutInt16** (const std::vector< int16_t > &vec)
- void PutInt16 (const std::vector< int > &vec, size_t num)

  Put operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void **PutInt16** (const std::vector< int > &vec)
- void PutInt16 (const std::valarray< int16_t > &vec, size_t num)

  Put operations for 'Int16' item type (signed 16-bit integers, 'SHORT')
- void **PutInt16** (const std::valarray< int16_t > &vec)
- void PutInt16 (const std::valarray< int > &vec, size_t num)

  Put operations for 'Int16' item type (signed 16-bit integers, 'SHORT')

- void **PutInt16** (const std::valarray< int > &vec)
- void PutUint16 (uint16_t us)

  Put operations for 'Uint16' item type (unsigned 16-bit integers)
- void PutUint16 (size_t us)

  The 'size_t' to uint16_t conversion gets extra checking since it is used in many vector length values and silently chopping off here would easily lead to data corruption.
- void **PutUint16** (const uint16_t *vec, size_t num)
- void PutUint16 (const std::vector< uint16_t > &vec, size_t num)

  Put operations for 'Uint16' item type (unsigned 16-bit integers)
- void **PutUint16** (const std::vector< uint16_t > &vec)
- void **PutUint16** (const std::vector< unsigned int > &vec, size_t num)
- void **PutUint16** (const std::vector< unsigned int > &vec)
- void PutUint16 (const std::valarray< uint16_t > &vec, size_t num)

  Put operations for 'Uint16' item type (unsigned 16-bit integers)
- void **PutUint16** (const std::valarray< uint16_t > &vec)
- void **PutUint16** (const std::valarray< unsigned int > &vec, size_t num)
- void **PutUint16** (const std::valarray< unsigned int > &vec)
- void PutInt32 (int32_t i)

  Put operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void **PutInt32** (const int32_t *vec, size_t num)
- void **PutInt32** (const long *vec, size_t num)
- void PutInt32 (const std::vector< int32_t > &vec, size_t num)

  Put operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void **PutInt32** (const std::vector< int32_t > &vec)
- void PutInt32 (const std::vector< long > &vec, size_t num)

  Put operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void **PutInt32** (const std::vector< long > &vec)
- void PutInt32 (const std::valarray< int32_t > &vec, size_t num)

  Put operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void **PutInt32** (const std::valarray< int32_t > &vec)
- void PutInt32 (const std::valarray< long > &vec, size_t num)

  Put operations for 'Int32' item type (signed 32-bit integers, 'LONG')
- void **PutInt32** (const std::valarray< long > &vec)
- void PutUint32 (uint32_t ui)

  Put operations for 'Uint32' item type (unsigned 32-bit integers)
- void **PutUint32** (const uint32_t *vec, size_t num)
- void PutUint32 (uint64_t us)

  The 'uint_t' to uint32_t conversion gets extra checking on machines with 64 bit integers since it is used in many vector length values (as a size_t) and silently chopping off here would easily lead to data corruption (well actually only with really huge data blocks).
- void PutUint32 (const std::vector< uint32_t > &vec, size_t num)

  Put operations for 'Uint32' item type (unsigned 32-bit integers)
- void **PutUint32** (const std::vector< uint32_t > &vec)
- void **PutUint32** (const std::vector< unsigned long > &vec, size_t num)
- void **PutUint32** (const std::vector< unsigned long > &vec)
- void PutUint32 (const std::valarray< uint32_t > &vec, size_t num)

  Put operations for 'Uint32' item type (unsigned 32-bit integers)
- void **PutUint32** (const std::valarray< uint32_t > &vec)
- void **PutUint32** (const std::valarray< unsigned long > &vec, size_t num)
- void **PutUint32** (const std::valarray< unsigned long > &vec)
- void PutInt64 (int64_t ll)

  Put operations for 'Int64' item type (signed 64-bit integers)
- void **PutInt64** (const int64_t *vec, size_t num)

- void PutInt64 (const std::vector< int64_t > &vec, size_t num)

  Put operations for 'Int64' item type (signed 64-bit integers)
- void **PutInt64** (const std::vector< int64_t > &vec)
- void PutInt64 (const std::valarray< int64_t > &vec, size_t num)

  Put operations for 'Int64' item type (signed 64-bit integers)
- void **PutInt64** (const std::valarray< int64_t > &vec)
- void PutUint64 (uint64_t ull)

  Put operations for 'Uint64' item type (unsigned 64-bit integers)
- void **PutUint64** (const uint64_t *vec, size_t num)
- void PutUint64 (const std::vector< uint64_t > &vec, size_t num)

  Put operations for 'Uint64' item type (unsigned 64-bit integers)
- void **PutUint64** (const std::vector< uint64_t > &vec)
- void PutUint64 (const std::valarray< uint64_t > &vec, size_t num)

  Put operations for 'Uint64' item type (unsigned 64-bit integers)
- void **PutUint64** (const std::valarray< uint64_t > &vec)
- void PutReal (float f)

  Put operations for 'Real' item type (32-bit IEEE float)
- void **PutReal** (const float *vec, size_t num)
- void PutReal (const std::vector< float > &vec, size_t num)

  Put operations for 'Real' item type (32-bit IEEE float)
- void **PutReal** (const std::vector< float > &vec)
- void PutReal (const std::valarray< float > &vec, size_t num)

  Put operations for 'Real' item type (32-bit IEEE float)
- void **PutReal** (const std::valarray< float > &vec)
- void **PutReal** (double d)
- void **PutReal** (const double *vec, size_t num)
- void PutReal (const std::vector< double > &vec, size_t num)

  Put operations for 'Real' item type (32-bit IEEE float)
- void **PutReal** (const std::vector< double > &vec)
- void PutReal (const std::valarray< double > &vec, size_t num)

  Put operations for 'Real' item type (32-bit IEEE float)
- void **PutReal** (const std::valarray< double > &vec)
- void PutDouble (double d)

  Put operations for 'Double' item type (64-bit IEEE double)
- void **PutDouble** (const double *vec, size_t num)
- void PutDouble (const std::vector< double > &vec, size_t num)

  Put operations for 'Double' item type (64-bit IEEE double)
- void **PutDouble** (const std::vector< double > &vec)
- void PutDouble (const std::valarray< double > &vec, size_t num)

  Put operations for 'Double' item type (64-bit IEEE double)
- void **PutDouble** (const std::valarray< double > &vec)
- void PutSfloat (double d)

  Put operations for 'Sfloat' item type (16-bit OpenGL float)
- void **PutSfloat** (const float *vec, size_t num)
- void **PutSfloat** (const double *vec, size_t num)
- void PutSfloat (const std::vector< float > &vec, size_t num)

  Put operations for 'Sfloat' item type (16-bit OpenGL float)
- void **PutSfloat** (const std::vector< double > &vec, size_t num)
- void **PutSfloat** (const std::vector< float > &vec)
- void **PutSfloat** (const std::vector< double > &vec)
- void PutSfloat (const std::valarray< float > &vec, size_t num)

  Put operations for 'Sfloat' item type (16-bit OpenGL float)

- void **PutSfloat** (const std::valarray< double > &vec, size_t num)
- void **PutSfloat** (const std::valarray< float > &vec)
- void **PutSfloat** (const std::valarray< double > &vec)
- void PutString (const char *s)

  Put operations for strings of any length.
- void **PutString16** (const char *s)
- void **PutString32** (const char *s)
- void PutString (const std::string &s)

  Put operations for strings of any length.
- void PutString16 (const std::string &s)

  Put operations for strings with old-style 16 bit length prefix.
- void PutString32 (const std::string &s)

  Put operations for strings with old-style 32 bit length prefix.
- void PutString (const std::vector< std::string > &vec, size_t num)

  Put operations for vectors of strings of any length.
- void **PutString** (const std::vector< std::string > &vec)
- void PutString (const std::valarray< std::string > &vec, size_t num)

  Put operations for valarrays of strings of any length.
- void **PutString** (const std::valarray< std::string > &vec)

Private Member Functions

- void **check_throw** (const char *op="unknown") const

Private Attributes

- IO_BUFFER * **iobuf**
- IO_ITEM_HEADER **item_header**
- int **rc**
- bool **put_flag**
- bool **throw_on_error**
- bool **done**
- bool active

  From {get|put}_item_begin to {get|put}_item_end.
- Item * **child**
- Item * **orig_parent**
- EventIO * **toplevel_parent**

Friends

- class **eventio::EventIO**

8.57.1 Detailed Description

This (sub-) class provides all the interfaces for putting and getting basic data to and from the embedded buffer.

8.57.2 Constructor & Destructor Documentation

8.57.2.1 eventio::EventIO::Item::Item (EventIO & *ev*, const char * *method*, size_t *type* = 0, size_t *version* = 0, long *ident* = 0, bool *user_flag* = false, bool *extended* = false)

Item constuctor for toplevel item takes the EventIO buffer as first argument.

There are three methods available here: a) "get" for reading from top-level of an I/O buffer, b) "put" for writing to an I/O level at top level (after finishing anything that may still be active in it currently), c) "append" for appending to an I/O buffer at its current position. If the I/O buffer has nothing active, this is identical to "put".

References active, _struct_IO_ITEM_HEADER::can_search, Done(), get_item_begin(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::level, put_item_begin_with_flags(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, and _struct_IO_ITEM_HEADER::version.

8.57.2.2 eventio::EventIO::Item::Item (EventIO::Item & *parent*, const char * *method*, size_t *type* = 0, size_t *version* = 0, long *ident* = 0, bool *user_flag* = false, bool *extended* = false)

Item constructor for sub-items takes the parent item as first argument.

There are two methods available here: a) "get" for reading from an I/O buffer at the level of the parent, after any active children are finished. b) "put" for writing to an I/O level at the level of the parent, after any active children are finished.

References active, _struct_IO_ITEM_HEADER::can_search, Done(), get_item_begin(), _struct_IO_ITEM_HEADER::ident, put_item_begin_with_flags(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, and _struct_IO_ITEM_HEADER::version.

8.57.2.3 eventio::EventIO::Item::Item (const Item & *item*)

The item copy constructor is something that might work to some extend, but sooner or later may result in corrupted data, in particular when multi-threaded programs try to write to the same file.

8.57.2.4 eventio::EventIO::Item::~Item (void)

Item destructor takes care that all sub-items are finished first.

In case of a toplevel item, it unregisters from the EventIO buffer.

8.57.3 Member Function Documentation

8.57.3.1 const char * eventio::EventIO::Item::Description (void)

Registered description for this type of I/O block.

Access to the registered description for this type of I/O block, if available.

References eventio_registered_description().

8.57.3.2 bool eventio::EventIO::Item::GetBool (void) [inline]

Individual 'bool' elements are stored as unsigned chars holding up to 8 bools.

Note that retrieving a vector of n bools is not the same as n times retrieving a single bool.

References GetUint8().

Referenced by read_test1(), read_test2(), and read_test3().

8.57.3.3 void eventio::EventIO::Item::GetCount (std::vector< uint16_t > & *vec*, size_t *num*)

Get operations for counts.

Get operations for counts (16 bit variant).

8.57.3.4 void eventio::EventIO::Item::GetCount (std::vector< uint16_t > & *vec*)

Get operations for counts.

Get operations for counts (16 bit variant).

8.57.3.5 void eventio::EventIO::Item::GetCount (std::valarray< uint16_t > & *vec*, size_t *num*)

Get operations for counts.

Get operations for counts (16 bit variant).

8.57.3.6 void eventio::EventIO::Item::GetCount (std::valarray< uint16_t > & *vec*)

Get operations for counts.

Get operations for counts (16 bit variant).

8.57.3.7 void eventio::EventIO::Item::GetInt16 (std::vector< int16_t > & *vec*, size_t *num*)

Get operations for signed counts.

Get operations for 'Int16' item type (signed 16-bit integers, 'SHORT')

8.57.3.8 void eventio::EventIO::Item::GetSCount (int16_t * *vec*, size_t *num*)

Get operations for counts.

Get operations for signed counts (16 bit variant)

8.57.3.9 void eventio::EventIO::Item::GetSCount (std::vector< int16_t > & *vec*, size_t *num*)

Get operations for signed counts.

Get operations for signed counts (16 bit variant).

8.57.3.10 void eventio::EventIO::Item::GetSCount (std::vector< int16_t > & *vec*)

Get operations for signed counts.

Get operations for signed counts (16 bit variant).

8.57.3.11 void eventio::EventIO::Item::GetSCount (std::valarray< int16_t > & *vec*, size_t *num*)

Get operations for signed counts.

Get operations for signed counts (16 bit variant).

8.57.3.12 void eventio::EventIO::Item::GetSCount (std::valarray< int16_t > & *vec*)

Get operations for signed counts.

Get operations for signed counts (16 bit variant).

8.57.3.13 int eventio::EventIO::Item::GetString (char * *s*, size_t *nmax*)

Get operations for strings with length encoded as Int16.

Get operation for C strings of any length.

Note: This is not the same as the old C method get_string(). To emulate that (e.g. for reading data written from old C code), use

```
*     size_t n = item.GetUint16();
*     item.GetUint8((const char *) s, n);
*
```

In the same way, the old C get_long_string() can be emulated by using a GetUint32() in the above code. The C language equivalent of the method here is called get_var_string().

References get_var_string().

Referenced by read_test1(), read_test2(), and read_test3().

8.57.3.14 uint8_t eventio::EventIO::Item::GetUint8 (void) [inline]

Get operations for bytes (only unsigned flavour implemented).

Even for std::vector and std::valarray the number of elements to get from the input buffer has to be specified explicitly.

Referenced by GetBool(), read_test1(), read_test2(), and read_test3().

8.57.3.15 int eventio::EventIO::Item::List (int *maxlevel* = 0, int *verbosity* = 0)

List item type, length, ID, ... + sub-items.

Show I/O block information to standard output.

References list_sub_items().

8.57.3.16 long eventio::EventIO::Item::NextSubItemIdent (void) const

Access to ID information of next sub-item at current reading position.

Returns

>= 0 (O.k., sub-item ident), -1 (error), -2 (end-of-buffer), -3 (no buffer)

References next_subitem_ident().

8.57.3.17 size_t eventio::EventIO::Item::NextSubItemLength (void) const

Access to length information of next sub-item at current reading position.

Returns

>= 0 (O.k., sub-item length), -1 (error), -2 (end-of-buffer), -3 (no buffer)

References next_subitem_length().

8.57.3.18 int eventio::EventIO::Item::NextSubItemType (void) const

Access to information of next sub-item at current reading position.

Access to type information of next sub-item at current reading position.

Returns

>= 0 (O.k., sub-item type), -1 (error), -2 (end-of-buffer), -3 (no buffer)

References next_subitem_type().

Referenced by read_test3().

8.57.3.19 void eventio::EventIO::Item::PutCount (const std::valarray< uint16_t > & *vec*, size_t *num*)

Put operations for counts.

Put operations for counts.

8.57.3.20 void eventio::EventIO::Item::PutInt16 (const std::vector< int16_t > & *vec*, size_t *num*)

Put operations for counts.

Put operations for 'Int16' item type (signed 16-bit integers, 'SHORT')

8.57.3.21 void eventio::EventIO::Item::PutSCount (const std::vector< int16_t > & *vec*, size_t *num*)

Put operations for counts.

Put operations for scounts.

8.57.3.22 void eventio::EventIO::Item::PutSCount (const std::valarray< int16_t > & *vec*, size_t *num*)

Put operations for scounts.

Put operations for scounts.

8.57.3.23 void eventio::EventIO::Item::PutString (const char * *s*)

Put operations for strings of any length.

(Note: encoding different from old C method put_string().)

Referenced by write_test1(), write_test2(), and write_test3().

8.57.3.24 void eventio::EventIO::Item::PutUint16 (size_t *us*) [inline]

The 'size_t' to uint16_t conversion gets extra checking since it is used in many vector length values and silently chopping off here would easily lead to data corruption.

References put_vector_of_uint16().

8.57.3.25 void eventio::EventIO::Item::PutUint32 (uint64_t *us*) [inline]

The 'uint_t' to uint32_t conversion gets extra checking on machines with 64 bit integers since it is used in many vector length values (as a size_t) and silently chopping off here would easily lead to data corruption (well actually only with really huge data blocks).

8.57.3.26 void eventio::EventIO::Item::PutUint8 (uint8_t *b*) [inline]

Put operations for bytes (only unsigned flavour implemented).

For std::vector and std::valarray sloppy variants are available to put all elements.

Referenced by write_test1(), write_test2(), and write_test3().

8.57.3.27 int eventio::EventIO::Item::Rewind (void)

Rewind to beginning of data area.

Rewind to beginning of the data area of the current item.

You can restart searching for specific sub-items again.

Returns

0 (ok), -1 (error)

References rewind_item().

Referenced by read_test3().

8.57.3.28 int eventio::EventIO::Item::Search (size_t *sub*)

Search for a specific sub-item.

Search the current item, starting at the current reading position, for the next sub-item of a specific type.

If the item is marked as searchable, it will check each sub-item (at the next level only) if it is of the desired type. That sub-item can then be immediately opened as a new Eventio::Item.

Parameters

sub	The type of sub-item that we are searching for.

Returns

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

References search_sub_item(), and _struct_IO_ITEM_HEADER::type.

Referenced by read_test3().

8.57.3.29 int eventio::EventIO::Item::Skip (void)

Skip a sub-item starting at the current position.

If the sub-item at the current position is of no further interest, you can skip it without having to decode its contents and go to the next item, if there is any.

Returns

0 (ok), -1 (error)

References skip_subitem().

8.57.3.30 const char * eventio::EventIO::Item::TypeName (void)

Registered name for this type of I/O block.

Access to the registered name for this type of I/O block, if available.

References eventio_registered_typename().

8.57.3.31 int eventio::EventIO::Item::Unget (void)

Completely undo getting this item, i.e.

rewind to beginning of its header. The item should not be used any more afterwards.

rewind to beginning of its header. The item should not be used any more afterwards.

Returns

0 (ok), -1 (error)

References unget_item().

8.57.3.32 int eventio::EventIO::Item::Unput (void)

Completely undo putting this item, as if never started.

The item should not be used any more afterwards.

The item should not be used any more afterwards.

Returns

0 (ok), -1 (error)

References unput_item().

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

- EventIO.hh
- EventIO.cc

8.58 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**

8.58.1 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

8.59 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 or 2)
- 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.

8.59.1 Detailed Description

Structure with per output telescope information keeping track of prerequisites.

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

- merge_simtel.c

8.60 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**

8.60.1 Detailed Description

Numbers to be summed up to obtain the moments.

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

- histogram.h

8.61 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**

8.61.1 Detailed Description

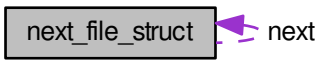
First, second, and higher moments of a 1-D histogram.

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

- histogram.h

8.62 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 files:

- read_hess.c
- read_hess_cc.cc

8.63 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.

8.63.1 Detailed Description

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

8.63.2 Field Documentation

8.63.2.1 double photo_electron::atime

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

8.63.2.2 int photo_electron::lambda

The wavelength of the photon.

8.63.2.3 int photo_electron::pixel

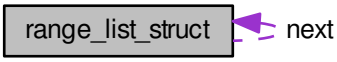
The pixel that was hit.

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

- mc_tel.h

8.64 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

8.65 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.

8.65.1 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.

8.65.2 Field Documentation

8.65.2.1 float* shower_extra_parameters::fparam

Space for extra floats, at least of size nfparam.

Referenced by clear_shower_extra_parameters(), and init_shower_extra_parameters().

8.65.2.2 long shower_extra_parameters::id

May identify to the user what the parameters should mean.

Referenced by clear_shower_extra_parameters(), and init_shower_extra_parameters().

8.65.2.3 int* shower_extra_parameters::iparam

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

Referenced by clear_shower_extra_parameters(), and init_shower_extra_parameters().

8.65.2.4 int shower_extra_parameters::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.

Referenced by clear_shower_extra_parameters(), init_shower_extra_parameters(), and write_hess_mc_shower().

8.65.2.5 size_t shower_extra_parameters::nfparam

Number of extra floating-point parameters.

Referenced by clear_shower_extra_parameters(), and init_shower_extra_parameters().

8.65.2.6 size_t shower_extra_parameters::niparam

Number of extra integer parameters.

Referenced by clear_shower_extra_parameters(), and init_shower_extra_parameters().

8.65.2.7 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.

Referenced by clear_shower_extra_parameters(), and init_shower_extra_parameters().

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

- mc_tel.h

8.66 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

8.67 telescope_list Struct Reference

Data Fields

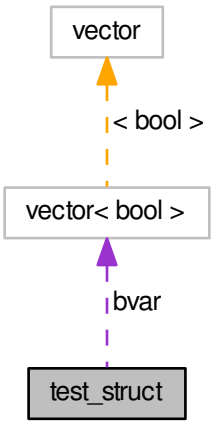
- size_t **min_tel**
- size_t **ntel**
- int * **tel_id**

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

- user_analysis.c

8.68 test_struct Struct Reference

Collaboration diagram for test_struct:



Data Fields

- long **lvar** [2]
- int **ilvar** [2]
- int **isvar** [2]
- short **svar** [3]
- double **fvar** [2]
- double **dvar** [2]
- size_t **nbvar**
- uint8_t **bvar** [2]
- int8_t **i8var** [2]
- uint8_t **u8var** [2]
- int16_t **i16var** [2]
- uint16_t **u16var** [2]
- int32_t **i32var** [2]
- uint32_t **u32var** [2]
- uint16_t **cnt16var** [4]
- int16_t **scnt16var** [10]
- uint32_t **cnt32var** [6]
- int32_t **scnt32var** [12]
- size_t **cntzvar** [6]
- ssize_t **scntzvar** [12]
- size_t **cntvar** [8]
- ssize_t **scntvar** [14]
- char **str16var** [10]
- char **str32var** [10]
- char **strvvar** [10]
- vector< bool > **bvar**
- int64_t **i64var** [2]

- uint64_t **u64var** [2]

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

- testio.c
- TestIO.cc

8.69 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

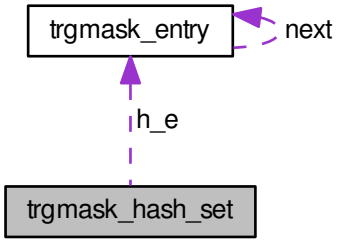
  Can be used in arrays but also in linked lists.

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

- io_trgmask.h

8.70 trgmask_hash_set Struct Reference

Collaboration diagram for trgmask_hash_set:



Data Fields

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

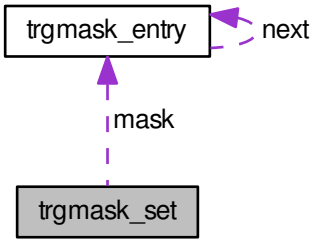
  Start of linked list for each possible hash value.

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

- io_trgmask.h

8.71 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

8.72 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 [2]
  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)
  } **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.

} **d**

8.72.1 Field Documentation

8.72.1.1 double user_parameters::calib_scale

Calibration scale from mean-p.e.

units to experimental units (0.0: like HESS).

Referenced by calibrate_amplitude(), and calibrate_pixel_amplitude().

8.72.1.2 double user_parameters::camera_clipping_deg

Pixel outside this radius (if > 0) should be ignored in image reconstruction.

Referenced by set_disabled_pixels(), and user_set_clipping().

8.72.1.3 double user_parameters::clip_amp

Pixel intensity clipped to this value after calibration, if this param is not zero.

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), reconstruct(), second_moments(), user_event_fill(), and user_set_clipamp().

8.72.1.4 double user_parameters::d_integ_param[2][4]

Integration-scheme- and gain-specific floating-point parameters.

8.72.1.5 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 .

Referenced by user_event_fill(), user_mc_event_fill(), and user_set_spectrum().

8.72.1.6 int user_parameters::integ_no_rescale

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

trace.

8.72.1.7 int user_parameters::integ_param[2]

Integration-scheme-specific integer parameters, typically:

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

Referenced by pixel_integration().

8.72.1.8 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.

Referenced by pixel_integration(), and reconstruct().

8.72.1.9 double user_parameters::min_amp

The minimum amplitude [peak p.e.

] of images usable for the analysis.

Referenced by main(), shower_reconstruct(), user_event_fill(), user_init(), and user_set_min_amp().

8.72.1.10 int user_parameters::min_pix

The minimum number of significant pixels in usable images.

Referenced by main(), user_event_fill(), user_init(), and user_set_min_pix().

8.72.1.11 int user_parameters::min_tel_img

Minimum and maximum number of usable images for events used in analysis.

Referenced by user_event_fill(), user_init(), and user_set_tel_img().

8.72.1.12 double user_parameters::tailcut_low

The lower and upper tail cuts for the standard two-level tail-cut scheme.

Referenced by main(), user_init(), and user_set_tail_cuts().

8.72.1.13 double user_parameters::theta_escale[4]

If the angular acceptance deviates from the 80% containment.

Referenced by user_event_fill(), and user_set_theta_escale().

8.72.1.14 int user_parameters::user_flags

1: HESS-style analysis standard cuts; 2: hard cuts; 3: loose cuts.

Referenced by user_event_fill(), user_init(), user_set_flags(), and user_set_max_theta().

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

- user_analysis.h

8.73 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**

8.73.1 Detailed Description

A struct used to store thread-specific data.

8.73.2 Field Documentation

8.73.2.1 const char* warn_specific_data::logfname

The name of the log file.

Used only when opening the file.

Referenced by set_log_file(), and warn_f_warning().

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

- warning.c

Chapter 9

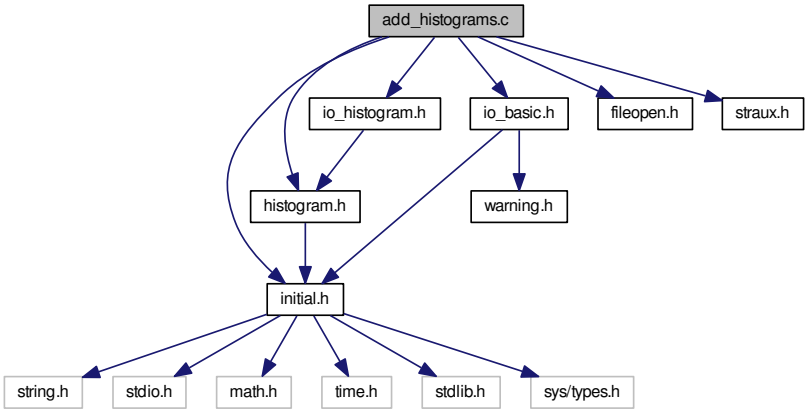
File Documentation

9.1 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.

9.1.1 Detailed Description

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

```
CVS $Date: 2014/06/24 14:29:40 $
```

Version

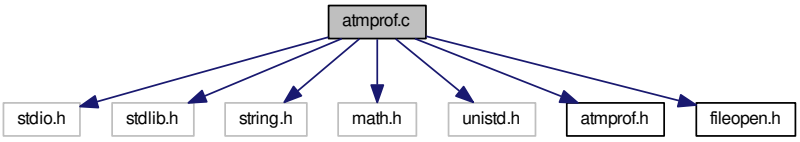
```
CVS $Revision: 1.2 $
```

9.2 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 "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.

- 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_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.

9.2.1 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.

The CORSIKA built-in profiles are not handled here.

Author

Konrad Bernloehr

Date

`CVS $Date: 2010/07/20 13:37:47 $`

Version

`CVS $Revision: 1.6 $`

9.2.2 Function Documentation

9.2.2.1 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]

References rpol().

9.2.2.2 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().

Referenced by user_event_fill().

9.2.2.3 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().

9.2.2.4 double refidx (double *height*)

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

Parameters

height	altitude [m]

Returns

index of refraction

References rpol().

Referenced by user_event_fill().

9.2.2.5 double rhofx (double *height*)

Density of the atmosphere as a function of altitude.

Parameters

height	altitude [m]

Returns

density [g/cm∗∗3]

References rpol().

9.2.2.6 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().

Referenced by heighx(), refidx(), rhofx(), and thickx().

9.2.2.7 double thickx (double *height*)

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

Parameters

height	altitude [m]

Returns

thickness [g/cm∗∗2]

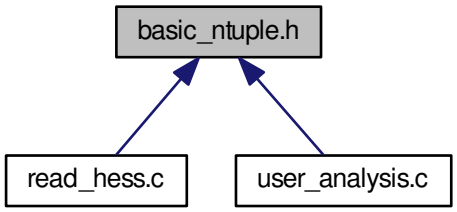
References rpol().

Referenced by user_event_fill().

9.3 basic_ntuple.h File Reference

Desclaration 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.

9.3.1 Detailed Description

Desclaration of the basic_ntuple struct.

9.3.2 Function Documentation

9.3.2.1 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.

References basic_ntuple::acceptance, basic_ntuple::alt, basic_ntuple::alt_true, basic_ntuple::az, basic_ntuple::az_true, basic_ntuple::chi2_e, basic_ntuple::event, basic_ntuple::lg_e, basic_ntuple::lg_e_true, basic_ntuple::mdisp, basic_ntuple::mscrl, basic_ntuple::mscrw, basic_ntuple::n_fail, basic_ntuple::n_img, basic_ntuple::n_pix, basic_ntuple::n_trg, basic_ntuple::n_tsl0, basic_ntuple::primary, basic_ntuple::rcm, basic_ntuple::run, basic_ntuple::sig_e, basic_ntuple::sig_mscrl, basic_ntuple::sig_mscrw, basic_ntuple::sig_theta, basic_ntuple::sig_xmax, basic_ntuple::theta, basic_ntuple::tslope, basic_ntuple::tsphere, basic_ntuple::weight, basic_ntuple::xc, basic_ntuple::xc_true, basic_ntuple::xfirst_true, basic_ntuple::xmax, basic_ntuple::xmax_true, basic_ntuple::yc, and basic_ntuple::yc_true.

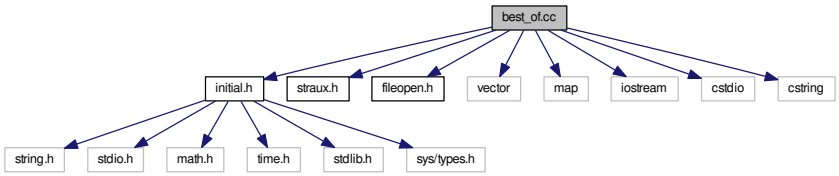
Referenced by main().

9.4 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_P-L_LGN2** = 4 }
- enum **BestChoice** {
  BestDiff =1, **BestIntegral** =2, **BestAngle** =3, **BestEres** =4,
  BestRate =5, **BestCombined** =6 }

Functions

- string **particle_type** (SpecType sp)
- double **Crab_Unit** (double E)
- static double **cu** (double x)
- 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)
- 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

9.4.1 Detailed Description

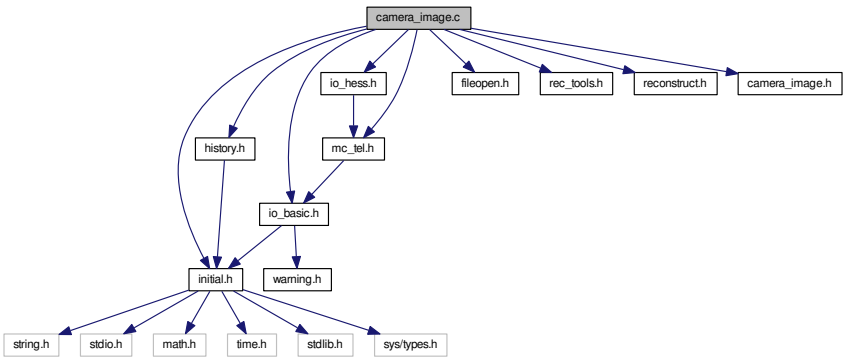
Tool for extracting best values from listings of 'rh3' sensitivity evaluations. Three versions of the 'rh3' output format are supported. All of the input (from standard input) should be in the same format type.

9.5 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 "history.h"
#include "io_hess.h"
#include "fileopen.h"
#include "rec_tools.h"
#include "reconstruct.h"
#include "camera_image.h"
```

Include dependency graph for camera_image.c:



Macros

- #define **H_MAX_NB1** 8
- #define **H_MAX_NB2** 24

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)

  Print a false-colour RGB value for a pixel intensity.
- 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 to a dedicated file.
- static int find_neighbours (CameraSettings *camset, int itel)

  Find the list of neighbours for each pixel.

Variables

- static char **ps_head1** []
- 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 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]

9.5.1 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

`CVS $Date: 2013/08/14 10:04:18 $`

Version

`CVS $Revision: 1.29 $`

9.5.2 Function Documentation

9.5.2.1 static int find_neighbours (CameraSettings * *camset,* int *itel*) [static]

Find the list of neighbours for each pixel.

References hess_camera_settings_struct::area, hess_camera_settings_struct::num_pixels, hess_camera_settings_struct::size, hess_camera_settings_struct::tel_id, hess_camera_settings_struct::xpix, and hess_camera_settings_struct::ypix.

9.5.2.2 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 to a dedicated file.

Also controlled via environment variables GAMMA_COEFF, GRAY_IMAGE, IMAGE_RANGE.

Parameters

image_fname	The name of the postscript image file.
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.
clip_amp,:	if >0, any calibrated amplitude is clipped not to exceed this value [mean p.e.].

References hess_tel_event_adc_struct::adc_sum, hess_shower_parameter::Alt, hess_mc_shower_struct::altitude, hess_tracking_event_data_struct::altitude_raw, hess_tel_image_struct::amplitude, angles_to_offset(), hess_shower_parameter::Az, hess_mc_shower_struct::azimuth, hess_tracking_event_data_struct::azimuth_raw, calibrate_pixel_amplitude(), hess_camera_settings_struct::cam_rot, hess_event_data_struct::central, hess_tel_event_adc_struct::data_red_mode, hess_mc_shower_struct::energy, fileclose(), fileopen(), hess_camera_settings_struct::flen, hess_central_event_data_struct::glob_count, H_MAX_TEL, HI_GAIN, hess_tel_event_data_struct::image_pixels, hess_tel_event_data_struct::img, hess_tel_event_adc_struct::known, hess_tel_image_struct::known, hess_tel_image_struct::l, LO_GAIN, hess_tel_event_data_struct::loc_count, hess_tel_event_data_struct::num_image_sets, hess_tel_monitor_struct::num_ped_slices, hess_camera_settings_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_tel_monitor_struct::pedestal, hess_tel_image_struct::phi, hess_pixel_list::pixel_list, hess_pixel_list::pixels, hess_tel_image_struct::pixels, hess_mc_shower_struct::primary_id, print_pix_col(), hess_tel_event_data_struct::raw, hess_run_header_struct::run, hess_event_data_struct::shower, hess_camera_settings_struct::size, hess_camera_settings_struct::tel_id, hess_run_header_struct::tel_pos, hess_event_data_struct::teldata, hess_event_data_struct::trackdata, hess_tel_event_data_struct::trigger_pixels, hess_tel_image_struct::w, hess_tel_image_struct::x, hess_mc_event_struct::xcore, hess_camera_settings_struct::xpix, hess_tel_image_struct::y, hess_mc_event_struct::ycore, hess_camera_settings_struct::ypix, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by main().

9.5.2.3 static void print_pix_col (double *n_o_r,* FILE * *psfile,* double *gamma_coeff*) [static]

Print a false-colour RGB value for a pixel intensity.

Referenced by hesscam_ps_plot().

9.5.3 Variable Documentation

9.5.3.1 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"
```

9.5.3.2 char ps_begin_page1[] [static]

Initial value:

```
=
"%%Page: "
```

9.5.3.3 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"
```

9.5.3.4 char ps_end_page[] [static]

Initial value:

```
=
"gr\n"
"showpage\n"
```

9.5.3.5 char ps_head1[] [static]

Initial value:

```
=
"%!PS-Adobe-2.0\n"


"%%Title: H.E.S.S. Telescope Simulation\n"

"%%Creator:"
```

9.5.3.6 char ps_trailer[] [static]

Initial value:

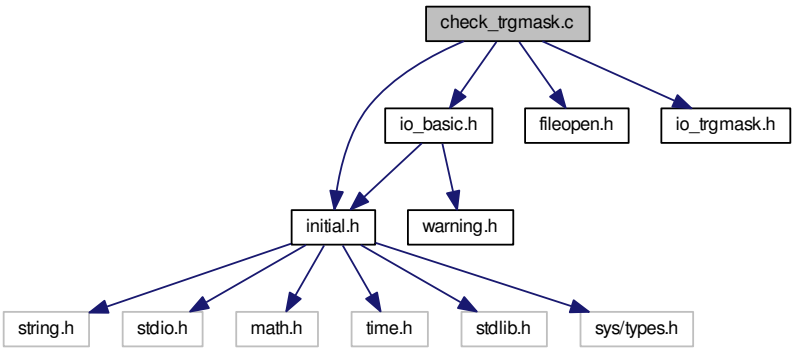
```
=
"rs\n"
```

9.6 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)

9.6.1 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
```

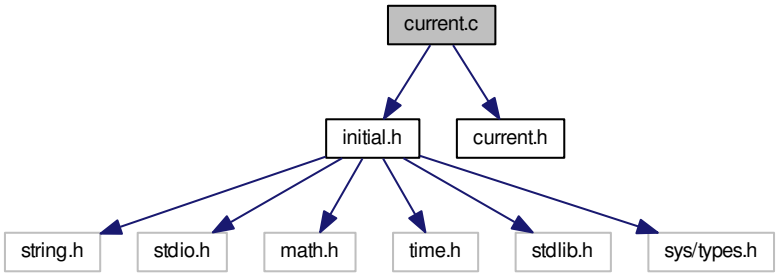
```
@author Konrad Bernloehr
```

9.7 current.c File Reference

Code to insert current time string into warnings.

```
#include "initial.h"
#include "current.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.
- 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

9.7.1 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, 2000, 2007

Date:

2010/07/20 13:37:45

Version

Revision:

1.7

9.7.2 Function Documentation

9.7.2.1 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.

References current_time(), and mkgmtime().

Referenced by time_string().

9.7.2.2 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 current_localtime().

9.7.2.3 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

Referenced by current_localtime().

9.7.2.4 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)

9.7.2.5 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)

9.7.2.6 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)

9.7.2.7 char* time_string (void)

Return a pointer to a formatted time-and-date string.

This string is reused (changed) on the next call.

Returns

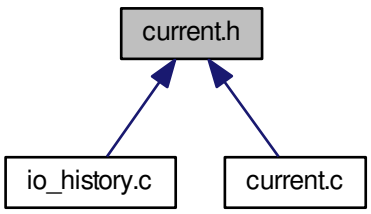
Time/date character string pointer.

References current_localtime().

9.8 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**

9.8.1 Detailed Description

Header file for optional current time add-on to warning.c.

Author

Konrad Bernloehr

Date

1993 (original version)

Date:

2010/07/20 13:37:45

Revision:

1.4

9.8.2 Function Documentation

9.8.2.1 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.

References current_time(), and mkgmtime().

Referenced by time_string().

9.8.2.2 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 current_localtime().

9.8.2.3 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

Referenced by current_localtime().

9.8.2.4 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)

9.8.2.5 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)

9.8.2.6 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)

9.8.2.7 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.

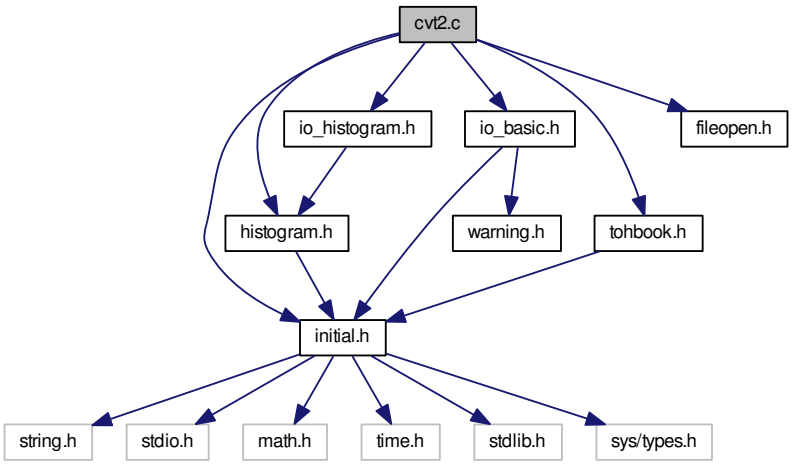
References current_localtime().

9.9 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.

9.9.1 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

`CVS $Date: 2014/02/20 10:53:06 $`

Version

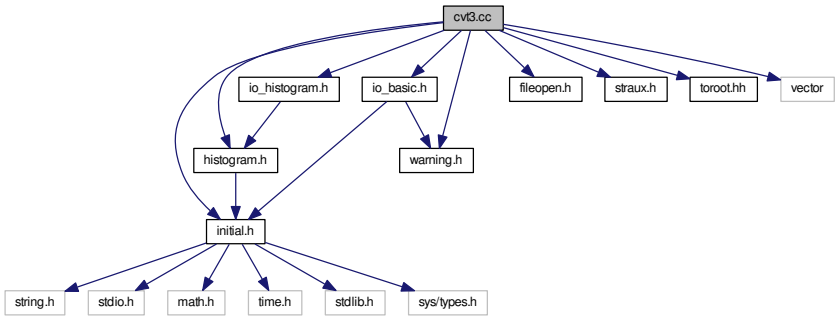
`CVS $Revision: 1.23 $`

9.10 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)

9.10.1 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

`CVS $Date: 2011/10/31 17:32:07 $`

Version

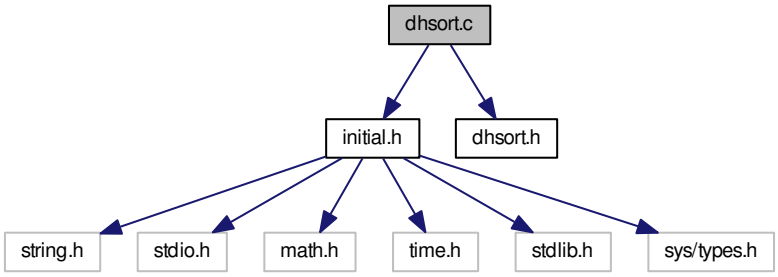
`CVS $Revision: 1.18 $`

9.11 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.

9.11.1 Detailed Description

dhsort - double type number heapsort

```
@author  Konrad Bernloehr
@date    $Date: 2010/07/20 13:37:45 $
@version $Revision: 1.6 $
```

```
            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]
```

9.11.2 Function Documentation

9.11.2.1 void dhsort (double * *dnum,* int *nel*)

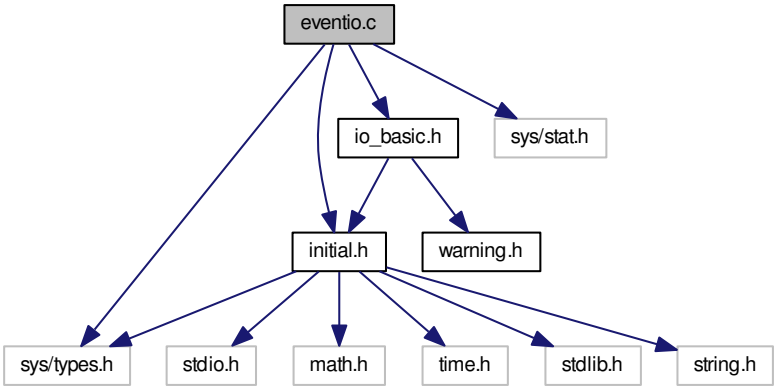
Perform a heap sort on a double array starting at dnum.

9.12 eventio.c File Reference

Basic functions for eventio data format.

```
#include "initial.h"
#include "io_basic.h"
#include <sys/types.h>
#include <sys/stat.h>
```

Include dependency graph for eventio.c:



Macros

- #define **NO_FOREIGN_PROTOTYPES** 1
- #define **IO_BUFFER_MINIMUM_SIZE** 32L
- #define **READ_BYTES**(fd, buf, nb)

Functions

- IO_BUFFER * allocate_io_buffer (size_t buflen)

  Dynamic allocation of an I/O buffer.
- int extend_io_buffer (IO_BUFFER *iobuf, unsigned next_byte, long increment)

  Extend the dynamically allocated I/O buffer.
- void free_io_buffer (IO_BUFFER *iobuf)

  Free an I/O buffer that has been allocated at run-time.
- void put_vector_of_byte (const BYTE *vec, int num, IO_BUFFER *iobuf)

  Put a vector of bytes into an I/O buffer.
- void get_vector_of_byte (BYTE *vec, int num, IO_BUFFER *iobuf)

  Get a vector of bytes from an I/O buffer.
- void put_count (uintmax_t n, IO_BUFFER *iobuf)

  Put an unsigned integer of unspecified length to an I/O buffer.
- void put_count32 (uint32_t n, IO_BUFFER *iobuf)

  Shortened version of put_count for up to 32 bits of data.
- void put_count16 (uint16_t n, IO_BUFFER *iobuf)

  Shortened version of put_count for up to 16 bits of data.
- uintmax_t get_count (IO_BUFFER *iobuf)

  Get an unsigned integer of unspecified length from an I/O buffer.
- uint32_t get_count32 (IO_BUFFER *iobuf)

  Get an unsigned 32 bit integer of unspecified length from an I/O buffer.
- uint16_t get_count16 (IO_BUFFER *iobuf)

  Get an unsigned 16 bit integer of unspecified length from an I/O buffer.
- void put_scount (intmax_t n, IO_BUFFER *iobuf)

  Put a signed integer of unspecified length to an I/O buffer.

- void put_scount32 (int32_t n, IO_BUFFER *iobuf)

  Shorter version of put_scount for up to 32 bytes of data.
- void put_scount16 (int16_t n, IO_BUFFER *iobuf)

  Shorter version of put_scount for up to 16 bytes of data.
- intmax_t get_scount (IO_BUFFER *iobuf)

  Get a signed integer of unspecified length from an I/O buffer.
- int16_t get_scount16 (IO_BUFFER *iobuf)

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

  Shortened version of get_scount for up to 32 bits of data.
- void put_vector_of_int_scount (const int *vec, int num, IO_BUFFER *iobuf)

  Put an array of ints as scount32 data into an I/O buffer.
- void get_vector_of_int_scount (int *vec, int num, IO_BUFFER *iobuf)

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

  Put a two-byte integer on an I/O buffer.
- void put_vector_of_short (const short *vec, int num, IO_BUFFER *iobuf)

  Put a vector of 2-byte integers on an I/O buffer.
- void put_vector_of_int (const int *vec, int num, IO_BUFFER *iobuf)

  Put a vector of integers (range -32768 to 32767) into I/O buffer.
- void put_vector_of_uint16 (const uint16_t *uval, int num, IO_BUFFER *iobuf)

  Put a vector of unsigned shorts into an I/O buffer.
- void get_vector_of_uint16 (uint16_t *uval, int num, IO_BUFFER *iobuf)

  Get a vector of unsigned shorts from an I/O buffer.
- uint16_t get_uint16 (IO_BUFFER *iobuf)

  Get one unsigned short from an I/O buffer.
- int get_short (IO_BUFFER *iobuf)

  Get a two-byte integer from an I/O buffer.
- void get_vector_of_short (short *vec, int num, IO_BUFFER *iobuf)

  Get a vector of short integers from I/O buffer.
- void get_vector_of_int (int *vec, int num, IO_BUFFER *iobuf)

  Get a vector of (small) integers from I/O buffer.
- void put_int32 (int32_t num, IO_BUFFER *iobuf)

  Write a four-byte integer to an I/O buffer.
- void put_vector_of_int32 (const int32_t *vec, int num, IO_BUFFER *iobuf)

  Put a vector of 32 bit integers into I/O buffer.
- int32_t get_int32 (IO_BUFFER *iobuf)

  Read a four byte integer from an I/O buffer.
- void get_vector_of_int32 (int32_t *vec, int num, IO_BUFFER *iobuf)

  Get a vector of 32 bit integers from I/O buffer.
- void put_uint32 (uint32_t num, IO_BUFFER *iobuf)

  Put a four-byte integer into an I/O buffer.
- void put_vector_of_uint32 (const uint32_t *vec, int num, IO_BUFFER *iobuf)

  Put a vector of 32 bit integers into I/O buffer.
- uint32_t get_uint32 (IO_BUFFER *iobuf)

  Get a four-byte unsigned integer from an I/O buffer.
- void get_vector_of_uint32 (uint32_t *vec, int num, IO_BUFFER *iobuf)

  Get a vector of 32 bit integers from I/O buffer.
- void put_long (long num, IO_BUFFER *iobuf)

  Put a four-byte integer taken from a 'long' into an I/O buffer.
- void put_vector_of_long (const long *vec, int num, IO_BUFFER *iobuf)

Put a vector of long int as 4-byte integers into an I/O buffer.

- long get_long (IO_BUFFER *iobuf)

  Get 4-byte integer from I/O buffer and return as a long int.
- void get_vector_of_long (long *vec, int num, IO_BUFFER *iobuf)

  Get a vector of 4-byte integers as long int from I/O buffer.
- int put_string (const char *s, IO_BUFFER *iobuf)

  Put a string of ASCII characters into an I/O buffer.
- int get_string (char *s, int nmax, IO_BUFFER *iobuf)

  Get a string of ASCII characters from an I/O buffer.
- int put_long_string (const char *s, IO_BUFFER *iobuf)

  Put a long string of ASCII characters into an I/O buffer.
- int get_long_string (char *s, int nmax, IO_BUFFER *iobuf)

  Get a long string of ASCII characters from an I/O buffer.
- int put_var_string (const char *s, IO_BUFFER *iobuf)

  Put a string of ASCII characters into an I/O buffer.
- int get_var_string (char *s, int nmax, IO_BUFFER *iobuf)

  Get a string of ASCII characters from an I/O buffer.
- void put_real (double dnum, IO_BUFFER *iobuf)

  Put a 4-byte floating point number into an I/O buffer.
- void put_vector_of_real (const double *dvec, int num, IO_BUFFER *iobuf)

  Put a vector of doubles as IEEE 'float' numbers into an I/O buffer.
- void put_vector_of_float (const float *fvec, int num, IO_BUFFER *iobuf)

  Put a vector of floats as IEEE 'float' numbers into an I/O buffer.
- double get_real (IO_BUFFER *iobuf)

  Get a floating point number (as written by put_real) from the I/O buffer.
- void get_vector_of_real (double *dvec, int num, IO_BUFFER *iobuf)

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

  Get a vector of floating point numbers as 'floats' from an I/O buffer.
- void dbl_to_sfloat (double dnum, uint16_t *snum)

  Convert a double to the internal representation of a 16 bit floating point number as specified in the OpenGL 3.1 standard.
- void put_sfloat (double dnum, IO_BUFFER *iobuf)

  Put a 16-bit float to an I/O buffer.
- double dbl_from_sfloat (const uint16_t *snum)

  Convert from the internal representation of an OpenGL 16-bit floating point number back to normal floating point representation.
- double get_sfloat (IO_BUFFER *iobuf)

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

  Begin putting another (sub-) item into the output buffer.
- int put_item_begin_with_flags (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, int user_flag, int extended)

  Begin putting another (sub-) item into the output buffer.
- int put_item_end (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  End of putting an item into the output buffer.
- int unput_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Undo writing at the present level.
- int get_item_begin (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Begin reading an item.
- int get_item_end (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

End reading an item.

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

  Go back to the beginning of an item being read.

- int next_subitem_type (IO_BUFFER *iobuf)

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

- long next_subitem_length (IO_BUFFER *iobuf)

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

- long next_subitem_ident (IO_BUFFER *iobuf)

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

- int skip_subitem (IO_BUFFER *iobuf)

  When the next sub-item is of no interest, it can be skipped.

- int search_sub_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, IO_ITEM_HEADER *sub_item_header)

  Search for an item of a specified type.

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

  Go back to the beginning of an item.

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

  Remove an item from an I/O buffer.

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

  Display the contents of sub-items on standard output.

- int reset_io_block (IO_BUFFER *iobuf)

  Reset an I/O block to its empty status.

- int write_io_block (IO_BUFFER *iobuf)

  Write an I/O block to the block's output.

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

  Find the beginning of the next I/O data block in the input.

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

  Read the data of an I/O block from the input.

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

  Skip the data of an I/O block from the input.

- int list_io_blocks (IO_BUFFER *iobuf, int verbosity)

  Show the top-level item of an I/O block on standard output.

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

  Copy a sub-item to another I/O buffer as top-level item.

- int append_io_block_as_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, BYTE *buffer, long length)

  Append data from one I/O block into another one.

- void set_eventio_registry_hook (struct ev_reg_entry *(*fptr)(unsigned long t))

  A single function for setting a registry search function is the interface between the eventio core code and any code implementing the registry.

- struct ev_reg_entry * find_ev_reg (unsigned long t)

  This optionally available function is implemented externally.

- const char * eventio_registered_typename (unsigned long type)

  Extract the name for a given type number, if available.

- const char * eventio_registered_description (unsigned long type)

  Extract the optional description for a given type number, if available.

Variables

- static EVREGSEARCH find_ev_reg_ptr

  We just keep a pointer to such a function in the core eventio code.
- static const char * none =""

  ----------— *find_ev_reg* ----------—

9.12.1 Detailed Description

Basic functions for eventio data format.

```
@author  Konrad Bernloehr
@date    1991 to 2010
@date    @verbatim CVS $Date: 2014/06/11 14:29:37 $
```

Version

```
    CVS $Revision: 1.41 $

================ General comments to eventio.c ======================

'eventio.c' provides an interface for an (almost) machine-independent
way to write and read event data, configuration data and Monte Carlo data.
Byte ordering of the data is unimportant and data written in both
byte orders are correctly read on any supported architecture.
Usually the data is written to/read from a file (or separate files for
different data types) to be opened before calling any eventio function.
Other ways to 'save' data (e.g. into memory or via dedicated networking
procedures can easily be incorporated by assigning an input and/or
output function to an I/O buffer instead of a file handle or pointer.
The data structure is designed to allow reading of a mixture of
different types of items from a single file. For this purpose, 'items'
(see below) should not be interspersed with low-level material and,
therefore, low-level functions should not be called from anywhere
outside eventio.c.

-------------------------------------------------------------------------

An 'item' has the following structure:

   Component    Type    Content      Description
   ---------    ----    -------      -----------
   sync-tag     long    0xD41F8A37   Signature of start of any item
                                     (only for top item, not for sub-items).
   type/version long    ...          Item type (bits 0 to 15), a single
                                     bit (16) of user data, extension flag
                                     (bit 17), reserved bits (18 to 19), and
                                     version of this item type (bits 20 to 31).
   ident        long    ...          Unique identification number of the
                                     item or -1.
   length       long    ...          No. of bytes following for this item
                                     (bits 0 to 29) and a flag indicating
                                     whether the item consists entirely of
                                     sub-items with known length (bit 30).
                                     Bit 31 must be 0. The bytes needed
                                     to pad the item to the next 4-byte
                                     boundary are included in the length.
 [ extension    long    ...          Only present if the extension flag in the
                                     type/version field is set. At this time,
                                     bits 0 to 11 will extend the length field
                                     (as bits 30 to 41). Note that 32-bit
                                     machines will not be able to take advantage
                                     of more than the one or two of those bits.
                                     Bits 12 to 31 are reserved and must be 0. ]
   data         ...     ...          Item data (may consist of elementary
                                     data and of sub-items)

Field 'sync-tag':
```

```
   The sync-tag is used to check that input is still synchronized.
   In the case of a synchronisation failure, all data should be skipped
   up to the next occurence of that byte combination or its reverse.
   The byte ordering of the sync-tag defines also the byte ordering
   of all data in the item. Only byte orders 0-1-2-3 and 3-2-1-0 are
   accepted at present.

Field 'type/version':
   This field consists of a type number in bits 0 to 15 (values
   0 to 65535), a single bit of user data (user flag), reserved bits
   17 to 19 (must be 0), and an item version number in bits 20 to 31
   (values 0 to 4095). Whenever the format of an item changes in a way
   which is incompatible with older reading software the version number
   has to be increased. This ways the reading software can be adapted
   to multiple format versions.

Field 'ident':
   Items of the same type can be distinguished if an identification
   number is supplied. Negative values are interpreted as 'no ident
   supplied'.

Field 'length':
   Each item and sub-item must have the number of bytes in its
   data area, including padding bytes, in bits 0 to 30 of this field.
   If an item consists entirely of sub-items and no atomic data, it can
   be searched for a specific type of sub-item without having to 'decode'
   (read from the buffer) any of the sub-items. Such an item is kind of
   a directory of sub-items and is marked by setting bit 30 of the
   length field on. The longest possible item length is thus (2^30 - 1).
   Note that the length field specifies the length of the rest of the
   item but not the sync-tag, type/version number, and length fields.
   All (sub-) items are padded to make the total length a multiple of 4
   bytes and the no. of padded bytes must be included in 'length'.

Field 'extension':
   This field is not present by default but requires the extension flag
   (as indicated in bit 17 of the 'type/version' field). Writing of
   data with the 'extension' flag can be forced by setting the
   'extended' element of the I/O buffer to 1 in advance. It can also
   be activated on a per-item basis but complications would arise once
   a sub-item goes beyond the 1 GB limit and any of its ancestors does
   not have the extension activated.
   Only bits 0-11 are used at this time (as bits 30-41 of the item length).
   On 32-bit systems only one or two of these bits would be usable
   (depending if lengths are counted with signed or unsigned integers).
   All other bits are reserved and must be set to 0 for now.
   Data written with the extension field will not be readable with
   pre-mid-2007 versions of eventio. Apart from that and within the
   limitations of the host architecture, reading of data with or
   without extension field is completely transparent to the application.

Data:
   Data of an item may be either sub-items or atomic data. An item may
   even consist of a mixture of both but in that case the sub-items
   are not accessible via 'directory' functions and can be processed
   only when the item data is 'decoded' by its corresponding 'read_...'
   function.
   The beginning of the data field is aligned on a 4-byte boundary to
   allow efficient access to data if the byte order needs not to be
   changed and if the data itself obeys the required alignment.

-------------------------------------------------------------------------

The 'atomic' data types are kept as close as possible to internal
data types. This data is only byte-aligned unless all atomic data
of an item obeys a 2-byte or 4-byte alignement.
Note that the ANSI C internal type int32_t typically corresponds to
both 'int' and 'long' on 32-bit machines but to 'int' only on
64-bit machines and to 'long' only on 16-bit systems.
Use the int32_t/uint32_t etc. types where the same length of
internal variables is required.
64-bit integer data are also implemented in eventio but not available
on all systems.
```

```
  Type    Int. type   Size (bytes)   Comments
  ----    ---------   ------------   --------
  byte    [u]int8_t   1              Character or very short integer.
  count   uintmax_t   1 to 9         Unsigned. Larger numbers need more bytes.
  scount  intmax_t    1 to 9         Signed. Larger numbers need more bytes.
  short   [u]int16_t  2              Short integer (signed or unsigned).
  long    [u]int32_t  4              Long integer (signed or unsigned).
  int64   [u]int64_t  8              Caution: not available on all systems.
  string  -           2+length       Preceded by 2-byte length of string.
  long str.  -        4+length       Preceded by 4-byte length of string.
  var str.   -        (1-5)+length   Preceded by length of string as 'count'.
  real    float       4              32-bit IEEE floating point number with
                                     the same byte order as a long integer.
  double  double      8              64-bit IEEE floating point number.
  sfloat     -        2              16-bit OpenGL floating point number

The byte-ordering of integers in input data is defined by that of
the sync-tag (magic number) preceding top-level items. Therefore,
the byte-ordering in a top-level item may differ from the ordering
in a previous item. For output data the default ordering is so far to
have the least-significant bytes first. This is the natural byte
order on Mips R3000 and higher (under Ultrix), DEC Alpha, VAX, and Intel
(80)x86 CPUs but the inverse of the natural byte order on Motorola 680x0,
RS6000, PowerPC, and Sparc CPUs. The ordering may change without
notice and without changing version numbers. Except for performance
considerations, the byte-ordering should not be relevant as long as
only the 0-1-2-3 and 3-2-1-0 orders are considered, and byte ordering
of floating point numbers is the same as for long integers.
Byte ordering for writing may be changed during run-time with the
'byte_order' element of the I/O buffer structure.
Note that on CPUs with non-IEEE floating point format like VAX writing
and reading of floating point numbers is likely to be less efficient
than on IEEE-format CPUs.

Note that if an 'int' variable is written via 'put_short()'
and then read again via 'get_short()' not only the
upper two bytes (on a 32-bit machine) are lost but
also the sign bit is propagated from bit 15 to the
upper 16 bits. Similarly, if a 'long' variable is written
via 'put_long()' and later read via 'get_long()' on a
64-bit-machine, not only the upper 4 bytes are lost but
also the sign in bit 31 is propagated to the upper 32 bits.

------------------------------------------------------------------------

Do not modify this file to include project-specific things!

========================================================================
```

9.12.2 Macro Definition Documentation

9.12.2.1 #define READ_BYTES(*fd, buf, nb*)

Value:

```
(((int)fd==0) ? \
  (ssize_t) fread((void *)buf,(size_t)1,(size_t)nb,stdin) : \
  read(fd,(void *)buf,(size_t)nb) )
```

9.12.3 Function Documentation

9.12.3.1 IO_BUFFER* allocate_io_buffer (size_t *buflen*)

Dynamic allocation of an I/O buffer.

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

Parameters

buflen	The length of the actual buffer in bytes. A safety margin of 4 bytes is added.

Returns

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

References _struct_IO_BUFFER::aux_count, _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, _struct_IO_BUFFER::extended, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::is_allocated, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_BUFFER::max_length, _struct_IO_BUFFER::min_length, _struct_IO_BUFFER::output_file, _struct_IO_BUFFER::output_fileno, _struct_IO_BUFFER::regular, _struct_IO_BUFFER::sub_item_length, _struct_IO_BUFFER::sync_err_count, _struct_IO_BUFFER::sync_err_max, _struct_IO_BUFFER::user_function, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::EventIO(), main(), and write_all_histograms().

9.12.3.2 int append_io_block_as_item (IO_BUFFER * *iobuf,* IO_ITEM_HEADER * *item_header,* BYTE * *buffer,* long *length*)

Append data from one I/O block into another one.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, extend_io_buffer(), _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::sub_item_length, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Append().

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

Copy a sub-item to another I/O buffer as top-level item.

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, reset_io_block(), _struct_IO_BUFFER::sub_item_length, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Copy(), and main().

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

Convert a double to the internal representation of a 16 bit floating point number as specified in the OpenGL 3.1 standard.

Parameters

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

Referenced by put_sfloat().

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

Extract the name for a given type number, if available.

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

References find_ev_reg(), ev_reg_entry::name, and none.

Referenced by list_io_blocks(), list_sub_items(), main(), and eventio::EventIO::Item::TypeName().

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

Extend the dynamically allocated I/O buffer.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::is_allocated, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::max_length, and _struct_IO_BUFFER::w_remaining.

Referenced by append_io_block_as_item(), copy_item_to_io_block(), put_int32(), put_item_begin_with_flags(), put_long(), put_short(), put_uint32(), put_vector_of_byte(), put_vector_of_uint16(), and read_io_block().

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

Find the beginning of the next I/O data block in the input.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, get_item_begin(), get_long(), get_uint32(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::sync_err_count, _struct_IO_BUFFER::sync_err_max, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_BUFFER::user_function, _struct_IO_ITEM_HEADER::version, and _struct_IO_BUFFER::w_remaining.

Referenced by check_autoload_trgmask(), eventio::EventIO::Find(), list_io_blocks(), and main().

9.12.3.8 void free_io_buffer (IO_BUFFER * *iobuf*)

Free an I/O buffer that has been allocated at run-time.

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

Parameters

iobuf	The buffer descriptor to be de-allocated.

Returns

(none)

References _struct_IO_BUFFER::buffer, and _struct_IO_BUFFER::is_allocated.

Referenced by write_all_histograms(), and eventio::EventIO::~EventIO().

9.12.3.9 uintmax_t get_count (IO_BUFFER * *iobuf*)

Get an unsigned integer of unspecified length from an I/O buffer.

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

Referenced by get_scount(), get_var_string(), print_trgmask(), read_test1(), read_test2(), read_test3(), and read_trgmask().

9.12.3.10 uint16_t get_count16 (IO_BUFFER * *iobuf*)

Get an unsigned 16 bit integer of unspecified length from an I/O buffer.

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

Referenced by get_scount16(), eventio::EventIO::Item::GetCount16(), and read_test1().

9.12.3.11 uint32_t get_count32 (IO_BUFFER * *iobuf*)

Get an unsigned 32 bit integer of unspecified length from an I/O buffer.

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

Referenced by get_scount32(), print_hess_centralevent(), read_hess_centralevent(), and read_test1().

9.12.3.12 int32_t get_int32 (IO_BUFFER * *iobuf*)

Read a four byte integer from an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_inquire_numbers(), config_binary_read_numbers(), get_long_string(), get_real(), eventio::EventIO::Item::GetInt32(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixelset(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_televt_head(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_televt_head(), and read_test1().

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

Begin reading an item.

Reads the header of an item.

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

Parameters

iobuf	The input buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, get_long(), get_uint32(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::length, _struct_IO_ITEM_HEADER::level, _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, _struct_IO_ITEM_HEADER::version, and _struct_IO_BUFFER::w_remaining.

Referenced by begin_read_tel_array(), config_binary_inquire_numbers(), config_binary_read_index(), config_binary_read_numbers(), config_binary_read_text(), config_binary_text_length(), find_io_block(), eventio::EventIO::Item::Item(), list_sub_items(), main(), next_subitem_ident(), next_subitem_length(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_

stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pixtime(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_array_end(), read_tel_array_head(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), read_test3(), read_trgmask(), search_sub_item(), and skip_subitem().

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

End reading an item.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::w_remaining.

Referenced by config_binary_read_index(), config_binary_read_numbers(), config_binary_read_text(), eventio::EventIO::Item::Done(), end_read_tel_array(), list_sub_items(), main(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_phot(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_phot(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pixtime(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_array_end(), read_tel_array_head(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), read_test3(), read_trgmask(), search_sub_item(), and skip_subitem().

9.12.3.15 long get_long (IO_BUFFER * *iobuf*)

Get 4-byte integer from I/O buffer and return as a long int.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_read_index(), find_io_block(), get_item_begin(), get_real(), get_time_blob(), next_subitem_type(), print_hess_runheader(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), read_hess_runheader(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), and read_test3().

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

Get a long string of ASCII characters from an I/O buffer.

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

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

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

References _struct_IO_BUFFER::data, get_int32(), and get_vector_of_byte().

Referenced by read_test1(), read_test2(), and read_test3().

9.12.3.17 double get_real (IO_BUFFER * *iobuf*)

Get a floating point number (as written by put_real) from the I/O buffer.

Parameters

iobuf	The I/O buffer descriptor;

Returns

The floating point number.

References get_int32(), and get_long().

Referenced by get_vector_of_float(), get_vector_of_real(), eventio::EventIO::Item::GetReal(), print_camera_layout(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixelset(), print_hess_pixtime(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), read_hess_camorgan(), read_hess_camsettings(), read_hess_centralevent(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixelset(), read_hess_pixtime(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_photo_electrons(), read_shower_longitudinal(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), and read_test1().

9.12.3.18 intmax_t get_scount (IO_BUFFER * *iobuf*)

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

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

References get_count().

Referenced by print_hess_camorgan(), print_hess_pixelset(), print_hess_teladc_samples(), read_hess_camorgan(), read_hess_pixelset(), read_hess_teladc_samples(), read_test1(), read_test2(), and read_test3().

9.12.3.19 int get_short (IO_BUFFER * *iobuf*)

Get a two-byte integer from an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_text_length(), get_string(), get_vector_of_int(), get_vector_of_short(), eventio::EventIO::Item::GetInt16(), eventio::EventIO::Item::GetUint16(), print_camera_layout(), print_hess_camorgan(), print_hess_centralevent(), print_hess_laser_calib(), print_hess_mc_pe_sum(), print_hess_mc_shower(), print_hess_pixel_list(), print_hess_pixtime(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televt_head(), print_hess_telimage(), print_histograms(), print_photo_electrons(), print_tel_photons(), read_camera_layout(), read_hess_camorgan(), read_hess_centralevent(), read_hess_laser_calib(), read_hess_mc_pe_sum(), read_hess_mc_shower(), read_hess_pixel_list(), read_hess_pixtime(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackset(), read_histograms_x(), read_photo_electrons(), read_shower_longitudinal(), read_tel_photons(), and read_test1().

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

Get a string of ASCII characters from an I/O buffer.

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

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

References _struct_IO_BUFFER::data, get_short(), and get_vector_of_byte().

Referenced by config_binary_read_text(), print_hess_runheader(), print_histograms(), read_hess_runheader(), read_histograms_x(), read_input_lines(), read_test1(), read_test2(), and read_test3().

9.12.3.21 uint16_t get_uint16 (IO_BUFFER * *iobuf*)

Get one unsigned short from an I/O buffer.

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

Parameters

iobuf	The output buffer descriptor.

Returns

The value obtained from the I/O buffer.

References get_vector_of_uint16().

Referenced by get_sfloat(), and print_hess_tel_monitor().

9.12.3.22 uint32_t get_uint32 (IO_BUFFER * *iobuf*)

Get a four-byte unsigned integer from an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by find_io_block(), get_item_begin(), eventio::EventIO::Item::GetUint32(), and read_test1().

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

Get a string of ASCII characters from an I/O buffer.

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

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

References _struct_IO_BUFFER::data, get_count(), and get_vector_of_byte().

Referenced by eventio::EventIO::Item::GetString(), read_test1(), read_test2(), and read_test3().

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

Get a vector of bytes from an I/O buffer.

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::data.

Referenced by config_binary_read_numbers(), get_long_string(), get_string(), get_var_string(), read_hess_tel_monitor(), read_test2(), and read_test3().

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

Get a vector of unsigned shorts from an I/O buffer.

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

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_read_numbers(), get_uint16(), print_hess_teladc_samples(), read_hess_tel_monitor(), read_hess_teladc_samples(), and read_hess_teladc_sums().

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

Show the top-level item of an I/O block on standard output.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::byte_order, eventio_registered_description(), eventio_registered_typename(), find_io_block(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, skip_io_block(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::user_flag, and _struct_IO_ITEM_HEADER::version.

Referenced by eventio::EventIO::List(), and main().

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

Display the contents of sub-items on standard output.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, eventio_registered_description(), eventio_registered_typename(), get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, list_sub_items(), search_sub_item(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, and _struct_IO_ITEM_HEADER::version.

Referenced by eventio::EventIO::Item::List(), list_sub_items(), and main().

9.12.3.28 long next_subitem_ident (IO_BUFFER * *iobuf*)

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

Parameters

iobuf	The input buffer descriptor.

Returns

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

References _struct_IO_BUFFER::data, get_item_begin(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and unget_item().

Referenced by eventio::EventIO::Item::NextSubItemIdent(), and read_hess_televent().

9.12.3.29 long next_subitem_length (IO_BUFFER * *iobuf*)

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

Parameters

iobuf	The input buffer descriptor.

Returns

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

References _struct_IO_BUFFER::data, get_item_begin(), _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_ITEM_HEADER::type, and unget_item().

Referenced by eventio::EventIO::Item::NextSubItemLength().

9.12.3.30 int next_subitem_type (IO_BUFFER * *iobuf*)

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

Parameters

iobuf	The input buffer descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, get_long(), _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, and _struct_IO_BUFFER::item_start_offset.

Referenced by config_binary_read_text(), config_binary_text_length(), eventio::EventIO::Item::NextSubItemType(), print_hess_event(), print_hess_mc_phot(), print_hess_televent(), read_hess_event(), read_hess_mc_phot(), read_hess_televent(), and read_test3().

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

Put an unsigned integer of unspecified length to an I/O buffer.

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

Parameters

n	The number to be saved. Even on systems with 64-bit integers, this must not exceed 2**32-1 with the current implementation.
iobuf	The output buffer descriptor.

Returns

(none)

References put_vector_of_byte().

Referenced by put_scount(), put_scount16(), put_scount32(), put_var_string(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Shortened version of put_count for up to 16 bits of data.

Returns

(none)

References put_vector_of_byte().

Referenced by eventio::EventIO::Item::PutCount16(), write_test1(), write_test2(), and write_test3().

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

Shortened version of put_count for up to 32 bits of data.

Returns

(none)

References put_vector_of_byte().

Referenced by write_hess_centralevent(), write_test1(), write_test2(), and write_test3().

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

Write a four-byte integer to an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_long_string(), put_real(), put_vector_of_int32(), eventio::EventIO::Item::PutInt32(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_televt_head(), and write_test1().

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

Begin putting another (sub-) item into the output buffer.

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

Parameters

iobuf	The output buffer descriptor.
item_header	The item header descriptor.

Returns

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

References put_item_begin_with_flags().

Referenced by begin_write_tel_array(), config_binary_envelope_begin(), config_binary_write_index(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(), write_hess_pointingcor(),

write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Begin putting another (sub-) item into the output buffer.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, extend_io_buffer(), _struct_IO_BUFFER::extended, _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::length, _struct_IO_ITEM_HEADER::level, put_long(), _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, _struct_IO_ITEM_HEADER::version, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Item::Item(), and put_item_begin().

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

End of putting an item into the output buffer.

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

Parameters

iobuf	The output buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::length, _struct_IO_ITEM_HEADER::level, put_uint32(), _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_BUFFER::w_remaining, and write_io_block().

Referenced by config_binary_envelope_end(), config_binary_write_index(), eventio::EventIO::Item::Done(), end_write_tel_array(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(),

write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Put a four-byte integer taken from a 'long' into an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by config_binary_write_index(), put_item_begin_with_flags(), put_real(), put_time_blob(), put_vector_of_long(), write_hess_runheader(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), and write_test3().

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

Put a long string of ASCII characters into an I/O buffer.

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

Parameters

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

Returns

Length of string

References put_int32(), put_short(), and put_vector_of_byte().

Referenced by write_test1(), write_test2(), and write_test3().

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

Put a 4-byte floating point number into an I/O buffer.

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

Parameters

dnum	The number to be put into the I/O buffer.

iobuf	The I/O buffer descriptor.

Returns

(none)

References put_int32(), and put_long().

Referenced by put_vector_of_float(), put_vector_of_real(), eventio::EventIO::Item::PutReal(), write_hess_camorgan(), write_hess_camsettings(), write_hess_centralevent(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixelset(), write_hess_pixtime(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_shower_longitudinal(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), and write_test1().

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

Put a signed integer of unspecified length to an I/O buffer.

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

Parameters

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

Returns

(none)

References put_count().

Referenced by write_hess_pixel_list(), write_hess_pixelset(), write_hess_teladc_samples(), write_test1(), write_test2(), and write_test3().

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

Shorter version of put_scount for up to 16 bytes of data.

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

Returns

(none)

References put_count().

Referenced by eventio::EventIO::Item::PutSCount16(), write_test1(), write_test2(), and write_test3().

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

Shorter version of put_scount for up to 32 bytes of data.

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

Returns

(none)

References put_count().

Referenced by put_vector_of_int_scount(), write_hess_camorgan(), write_hess_pixtime(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_televt_head(), write_hess_telimage(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Put a two-byte integer on an I/O buffer.

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

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_long_string(), put_string(), put_vector_of_int(), put_vector_of_short(), write_camera_layout(), write_hess_camorgan(), write_hess_centralevent(), write_hess_laser_calib(), write_hess_mc_pe_sum(), write_hess_mc_shower(), write_hess_pixel_list(), write_hess_pixtime(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackset(), write_histograms(), write_photo_electrons(), write_shower_longitudinal(), write_tel_compact_photons(), write_tel_photons(), and write_test1().

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

Put a string of ASCII characters into an I/O buffer.

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

Parameters

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

Returns

Length of string

References put_short(), and put_vector_of_byte().

Referenced by write_hess_runheader(), write_histograms(), write_input_lines(), write_test1(), write_test2(), and write_test3().

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

Put a four-byte integer into an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_item_end(), put_vector_of_uint32(), eventio::EventIO::Item::PutUint32(), remove_item(), and write_test1().

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

Put a string of ASCII characters into an I/O buffer.

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

Parameters

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

Returns

Length of string

References put_count(), and put_vector_of_byte().

Referenced by write_test1(), write_test2(), and write_test3().

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

Put a vector of bytes into an I/O buffer.

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_count(), put_count16(), put_count32(), put_long_string(), put_string(), put_var_string(), write_hess_tel_monitor(), write_test2(), and write_test3().

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

Put a vector of integers (range -32768 to 32767) into I/O buffer.

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

References put_short().

Referenced by write_hess_camorgan(), write_hess_centralevent(), write_hess_mc_pe_sum(), write_hess_pixel_list(), write_hess_pixelset(), write_hess_pixtime(), write_hess_runheader(), write_hess_shower(), write_hess_teladc_sums(), write_hess_telimage(), write_test2(), and write_test3().

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

Put a vector of 2-byte integers on an I/O buffer.

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

References put_short().

Referenced by write_hess_tel_monitor(), write_test2(), and write_test3().

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

Put a vector of unsigned shorts into an I/O buffer.

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

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_sfloat(), eventio::EventIO::Item::PutUint16(), write_hess_tel_monitor(), write_hess_teladc_samples(), and write_hess_teladc_sums().

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

Read the data of an I/O block from the input.

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

Parameters

iobuf	The I/O buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data_pending, extend_io_buffer(), _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, skip_io_block(), _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::user_function.

Referenced by check_autoload_trgmask(), main(), and eventio::EventIO::Read().

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

Remove an item from an I/O buffer.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, put_uint32(), _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, and _struct_IO_BUFFER::w_remaining.

9.12.3.54 int reset_io_block (IO_BUFFER * *iobuf*)

Reset an I/O block to its empty status.

Parameters

iobuf	The I/O buffer descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::min_length, _struct_IO_BUFFER::regular, _struct_IO_BUFFER::sub_item_length, and _struct_IO_BUFFER::w_remaining.

Referenced by copy_item_to_io_block(), and main().

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

Go back to the beginning of an item.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, and _struct_IO_BUFFER::w_remaining.

Referenced by read_test3(), and eventio::EventIO::Item::Rewind().

9.12.3.56 int search_sub_item (IO_BUFFER * *iobuf,* IO_ITEM_HEADER * *item_header,* IO_ITEM_HEADER * *sub_item_header*)

Search for an item of a specified type.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, get_item_begin(), get_item_end(), _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::w_remaining.

Referenced by list_sub_items(), read_test3(), and eventio::EventIO::Item::Search().

9.12.3.57 void set_eventio_registry_hook (struct ev_reg_entry *(*)(unsigned long t) *fptr*)

A single function for setting a registry search function is the interface between the eventio core code and any code implementing the registry.

This search function is also responsible for initializing the registry. By default, no such registry is used.

Parameters

fptr	A pointer to the registry search function.

References find_ev_reg().

Referenced by set_ev_reg_std().

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

Skip the data of an I/O block from the input.

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

Parameters

iobuf	The I/O buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data_pending, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::regular, _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::user_function.

Referenced by list_io_blocks(), main(), read_io_block(), and eventio::EventIO::Skip().

9.12.3.59 int skip_subitem (IO_BUFFER * *iobuf*)

When the next sub-item is of no interest, it can be skipped.

Parameters

iobuf	I/O buffer descriptor.

Returns

0 (ok), -1 (error)

References get_item_begin(), get_item_end(), and _struct_IO_ITEM_HEADER::type.

Referenced by print_hess_event(), print_hess_mc_phot(), print_hess_televent(), read_hess_mc_phot(), read_hess_televent(), and eventio::EventIO::Item::Skip().

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

Go back to the beginning of an item being read.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, and _struct_IO_BUFFER::w_remaining.

Referenced by config_binary_inquire_numbers(), config_binary_read_numbers(), config_binary_text_length(), next_subitem_ident(), next_subitem_length(), read_photo_electrons(), read_tel_photons(), and eventio::EventIO::Item::Unget().

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

Undo writing at the present level.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, _struct_IO_ITEM_HEADER::use_extension, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Item::Unput(), write_hess_event(), and write_hess_televent().

9.12.3.62 int write_io_block (IO_BUFFER * *iobuf*)

Write an I/O block to the block's output.

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

Parameters

iobuf	The I/O buffer descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::output_file, _struct_IO_BUFFER::output_fileno, _struct_IO_BUFFER::user_function, and _struct_IO_BUFFER::w_remaining.

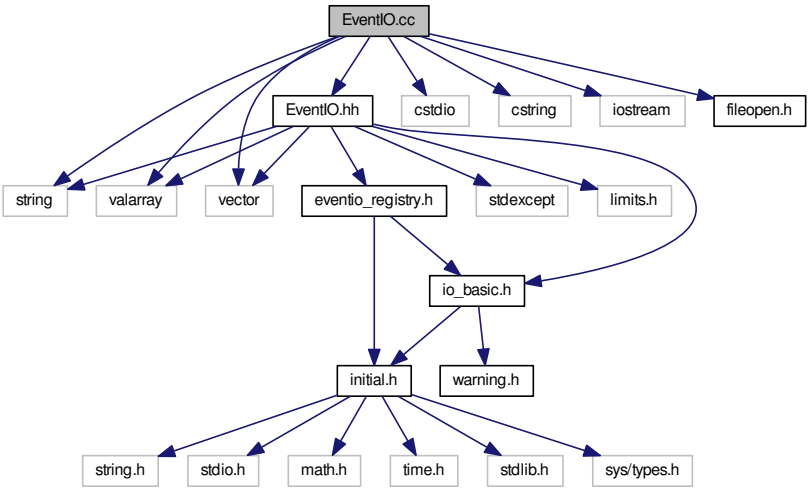
Referenced by main(), put_item_end(), eventio::EventIO::Write(), and write_io_block_to_file().

9.13 EventIO.cc File Reference

Implementation of methods for the C++ interface to the eventio data format.

```
#include <string>
#include <valarray>
#include <vector>
#include <cstdio>
#include <cstring>
#include <iostream>
#include "fileopen.h"
#include "EventIO.hh"
```

Include dependency graph for EventIO.cc:



Namespaces

- eventio

The classes of this interface belong to the namespace "eventio".

9.13.1 Detailed Description

Implementation of methods for the C++ interface to the eventio data format.

Author

Konrad Bernloehr

Date

Initial release: April 2003

Date:

2014/05/30 15:31:44

Revision:

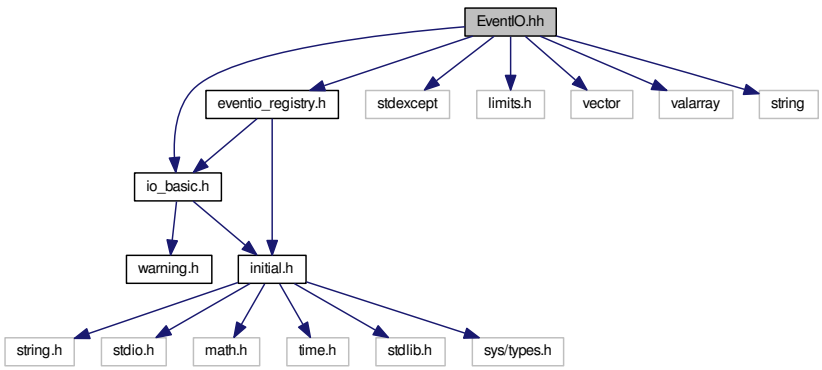
1.32

9.14 EventIO.hh File Reference

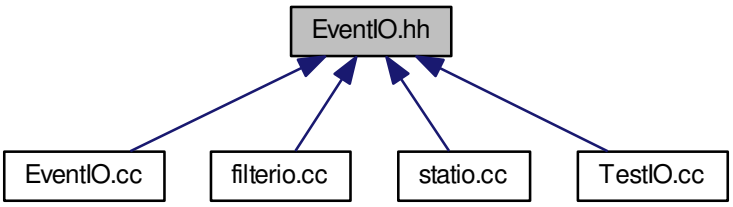
C++ interface to the eventio data format.

```
#include "io_basic.h"
#include "eventio_registry.h"
#include <stdexcept>
#include <limits.h>
#include <vector>
#include <valarray>
#include <string>
```

Include dependency graph for EventIO.hh:



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



Data Structures

- class eventio::EventIO

  This class provides the embedded buffer, the file I/O interface and the toplevel item access.
- class eventio::EventIO::Item

  This (sub-) class provides all the interfaces for putting and getting basic data to and from the embedded buffer.

Namespaces

- eventio

  The classes of this interface belong to the namespace "eventio".

Macros

- #define **HAVE_64BIT_INT** 1
- #define **HAVE_STD_VECTOR** 1
- #define **HAVE_STD_VALARRAY** 1
- #define **HAVE_STD_STRING**

Typedefs

- typedef class EventIO::Item **eventio::EventIO_Item**

9.14.1 Detailed Description

C++ interface to the eventio data format.

Author

Konrad Bernloehr

Date

Initial release: April 2003

Date:

2014/05/30 15:31:44

Revision:

1.41

Header file for C++ interface to the eventio data format. In constrast to the C interface, the C++ interface can take care of the completion of "put" and "get" operations at the end of the lifetime of an object. Nested data objects are much more straight-forward than in C.

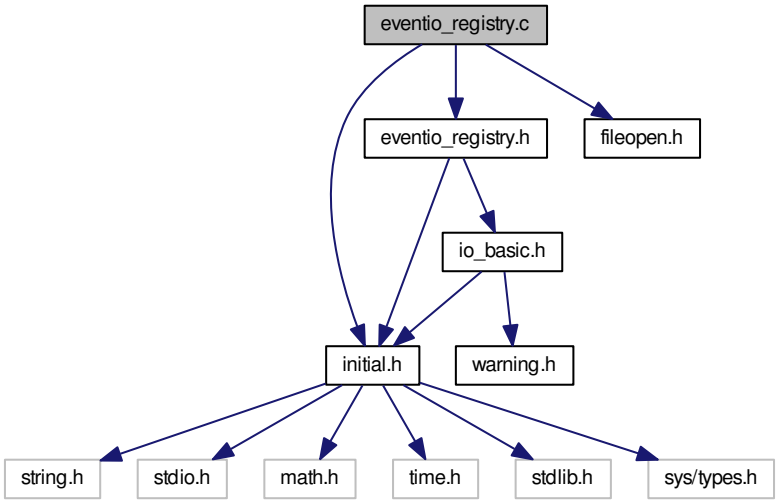
Overloading of methods is used to have the same name for all "put" and "get" operations of the same basic data type. The operations are optionally available also for std::vector and std::valarray arguments of (selected) suitable types - but only if the includes were done before this file is included. For standards conforming compilers the macros _CPP_VECTOR, _CPP_VALARRAY, and _CPP_STRING are used as indicators. For gcc 2.96, the macros **SGI_STL_VECTOR, __STD_VALARRAY**, and **BASTRING** seem to be corresponding indicators. For GCC version 3.4 this was changed to _GLIBCXX_VECTOR, _GLIBCXX_VALARRAY, and _GLIBCXX_STRING.

9.15 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

9.15.1 Detailed Description

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

Author

Konrad Bernloehr

Date

2014
CVS

Date:

2014/06/03 16:19:44

Version

CVS

Revision:

1.3

9.15.2 Function Documentation

9.15.2.1 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.

References ev_reg_chain::entry, read_default_registry(), and ev_reg_entry::type.

Referenced by set_ev_reg_std().

9.15.2.2 int read_eventio_registry (const char * *fname*)

Read the type names and descriptions into the registry.

Note: this will only be done once.

References ev_reg_chain::entry, fileclose(), fileopen(), new_reg_entry(), and ev_reg_entry::type.

Referenced by read_default_registry().

9.15.2.3 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(), and set_eventio_registry_hook().

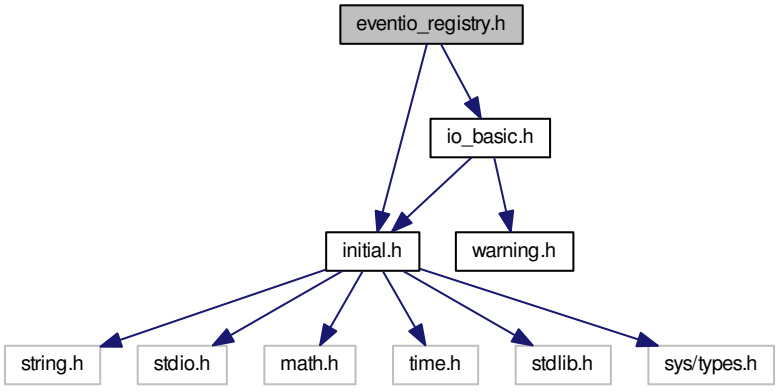
Referenced by main().

9.16 eventio_registry.h File Reference

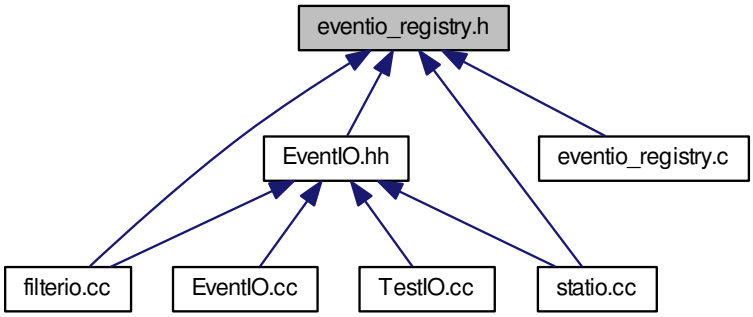
Register and enquire about well-known I/O block types.

```
#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.

9.16.1 Detailed Description

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

Author

Konrad Bernloehr

Date

2014
CVS

Date:

2014/06/01 11:33:04

Version

CVS

Revision:

1.2

9.16.2 Function Documentation

9.16.2.1 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.

References ev_reg_chain::entry, read_default_registry(), and ev_reg_entry::type.

Referenced by set_ev_reg_std().

9.16.2.2 int read_eventio_registry (const char * *fname*)

Read the type names and descriptions into the registry.

Note: this will only be done once.

References ev_reg_chain::entry, fileclose(), fileopen(), new_reg_entry(), and ev_reg_entry::type.

Referenced by read_default_registry().

9.16.2.3 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(), and set_eventio_registry_hook().

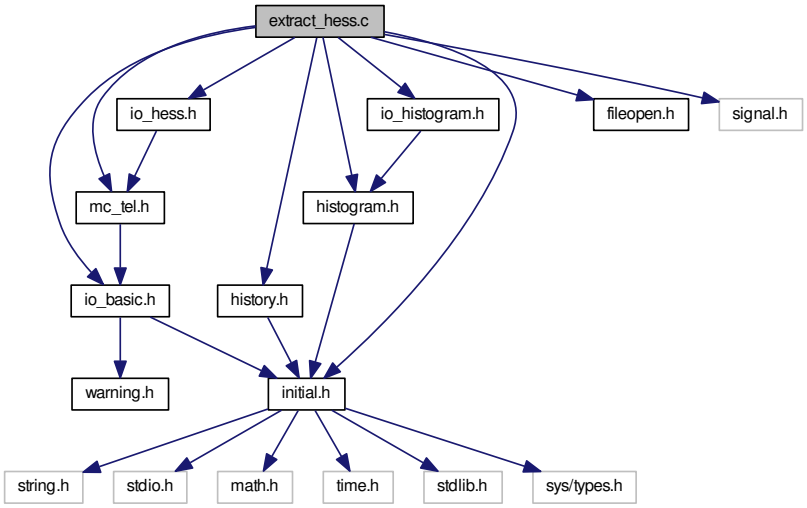
Referenced by main().

9.17 extract_hess.c File Reference

Extract part of the H.E.S.S.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "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**

9.17.1 Detailed Description

Extract part of the H.E.S.S. data from sim_hessarray.

Author

Konrad Bernloehr

Date

CVS $Date: 2011/04/15 13:48:03 $

Version

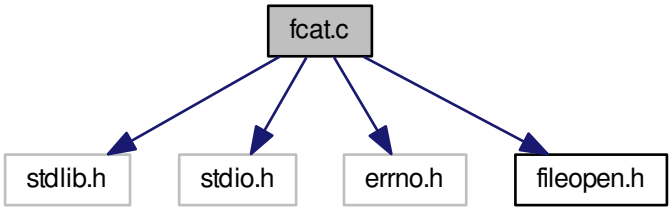
CVS $Revision: 1.6 $

9.18 fcat.c File Reference

Trivial test and utility program for the fileopen/fileclose functions.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include "fileopen.h"
```

Include dependency graph for fcat.c:



Macros

- #define **BSIZE** 8192

Functions

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

9.18.1 Detailed Description

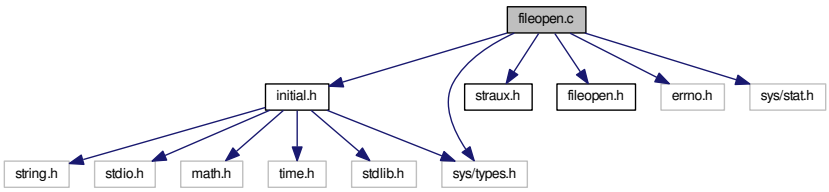
Trivial test and utility program for the fileopen/fileclose functions.

9.19 fileopen.c File Reference

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

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

Include dependency graph for fileopen.c:



Data Structures

- struct incpath

  An element in a linked list of include paths.

Functions

- 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.
- 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 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.

9.19.1 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) 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`:filepath' where filepath can be an absolute path (starting with `'/'`) or one relative to the users home directory.
- Input and output can also be from/to a user-defined program. Restrictions apply there which prevent execution of any program by default. Either a list of accepted execution paths has to be set up beforehand with initexepath()/addexepath() or permissive mode can be enabled, allowing execution of any given program.

Author

Konrad Bernloehr

Date

Nov. 2000

`CVS $Date: 2014/06/23 09:34:45 $`

Version

`CVS $Revision: 1.15 $`

9.19.2 Function Documentation

9.19.2.1 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.

9.19.2.2 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 getword(), incpath::next, incpath::path, root_path, and verbose.

Referenced by addexepath(), and initpath().

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

Helper function for opening a compressed file through a fifo.

References verbose.

Referenced by fileopen().

9.19.2.4 void disable_permissive_pipes (void)

Disable the permissive execution of pipes.

References permissive_pipes.

Referenced by set_permissive_pipes().

9.19.2.5 void enable_permissive_pipes (void)

Enable the permissive execution of pipes.

References permissive_pipes.

Referenced by set_permissive_pipes().

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

Helper function for opening a pipe from or to a given program.

References incpath::next, incpath::path, permissive_pipes, and verbose.

Referenced by fileopen().

9.19.2.7 int fileclose (FILE * *f*)

Close a file or fifo but not if it is one of the standard streams.

Referenced by check_autoload_trgmask(), eventio::EventIO::CloseInput(), eventio::EventIO::CloseOutput(), hesscam_ps_plot(), main(), read_eventio_registry(), trgmask_scan_log(), write_all_histograms(), write_tel_compact_photons(), and write_tel_photons().

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

Search for a file in the include path list and open it if possible.

References cmp_popen(), exe_popen(), initpath(), incpath::next, incpath::path, root_path, ssh_popen(), uri_popen(), and verbose.

Referenced by check_autoload_trgmask(), hesscam_ps_plot(), init_atmprof(), main(), eventio::EventIO::OpenInput(), eventio::EventIO::OpenOutput(), read_eventio_registry(), trgmask_scan_log(), write_all_histograms(), write_tel_compact_photons(), and write_tel_photons().

9.19.2.9 static void freeexepath ( ) [static]

Free a whole list of execution path elements.

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

9.19.2.10 static void freepath ( ) [static]

Free a whole list of include path elements.

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

Referenced by initpath().

9.19.2.11 void initpath (const char * *default_path*)

Init the path list, with default_path as the only entry.

References addpath(), freepath(), getword(), and verbose.

Referenced by fileopen().

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

Show the list of include paths.

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

9.19.2.13 void set_permissive_pipes (int *p*)

Enable or disable the permissive execution of pipes.

References disable_permissive_pipes(), and enable_permissive_pipes().

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

Helper function for opening a file with a URI (http:// etc.).

References verbose.

Referenced by fileopen().

9.19.3 Variable Documentation

9.19.3.1 int permissive_pipes = 0 [static]

Allow any execution pipe command if this variable is non-zero.

Referenced by disable_permissive_pipes(), enable_permissive_pipes(), and exe_popen().

9.19.3.2 struct incpath* root_exe_path = NULL [static]

The starting element for execution paths.

Referenced by addexepath().

9.19.3.3 struct incpath* root_path = NULL [static]

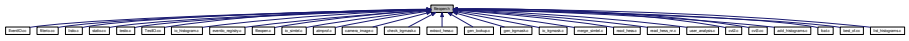
The starting element of include paths.

Referenced by addexepath(), addpath(), and fileopen().

9.20 fileopen.h File Reference

Function prototypes for fileopen.c.

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



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.
- 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.

9.20.1 Detailed Description

Function prototypes for fileopen.c.

Author

Konrad Bernloehr

Date

CVS $Date: 2014/06/23 09:34:45 $

Version

CVS $Revision: 1.7 $

9.20.2 Function Documentation

9.20.2.1 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.

9.20.2.2 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 getword(), incpath::next, incpath::path, root_path, and verbose.

Referenced by addexepath(), and initpath().

9.20.2.3 void disable_permissive_pipes (void)

Disable the permissive execution of pipes.

References permissive_pipes.

Referenced by set_permissive_pipes().

9.20.2.4 void enable_permissive_pipes (void)

Enable the permissive execution of pipes.

References permissive_pipes.

Referenced by set_permissive_pipes().

9.20.2.5 int fileclose (FILE * *f*)

Close a file or fifo but not if it is one of the standard streams.

Referenced by check_autoload_trgmask(), eventio::EventIO::CloseInput(), eventio::EventIO::CloseOutput(), hesscam_ps_plot(), main(), read_eventio_registry(), trgmask_scan_log(), write_all_histograms(), write_tel_compact_photons(), and write_tel_photons().

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

Search for a file in the include path list and open it if possible.

References cmp_popen(), exe_popen(), initpath(), incpath::next, incpath::path, root_path, ssh_popen(), uri_popen(), and verbose.

Referenced by check_autoload_trgmask(), hesscam_ps_plot(), init_atmprof(), main(), eventio::EventIO::OpenInput(), eventio::EventIO::OpenOutput(), read_eventio_registry(), trgmask_scan_log(), write_all_histograms(), write_tel_compact_photons(), and write_tel_photons().

9.20.2.7 void initpath (const char * *default_path*)

Init the path list, with default_path as the only entry.

References addpath(), freepath(), getword(), and verbose.

Referenced by fileopen().

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

Show the list of include paths.

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

9.20.2.9 void set_permissive_pipes (int *p*)

Enable or disable the permissive execution of pipes.

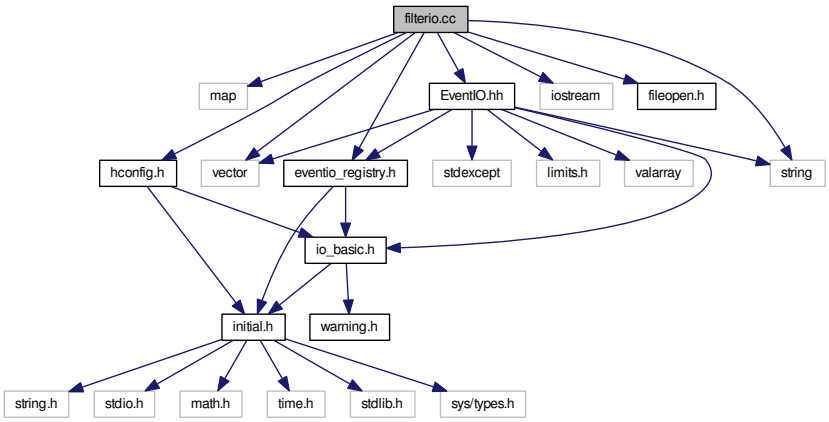
References disable_permissive_pipes(), and enable_permissive_pipes().

9.21 filterio.cc File Reference

A program for filtering eventio data blocks.

```
#include <map>
#include <vector>
#include "EventIO.hh"
#include <iostream>
#include <string>
#include <fileopen.h>
#include <hconfig.h>
#include <eventio_registry.h>
```

Include dependency graph for filterio.cc:



Data Structures

- struct iostats

Macros

- #define **__STDC_LIMIT_MACROS** 1

Functions

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

  Main program.

9.21.1 Detailed Description

A program for filtering eventio data blocks.

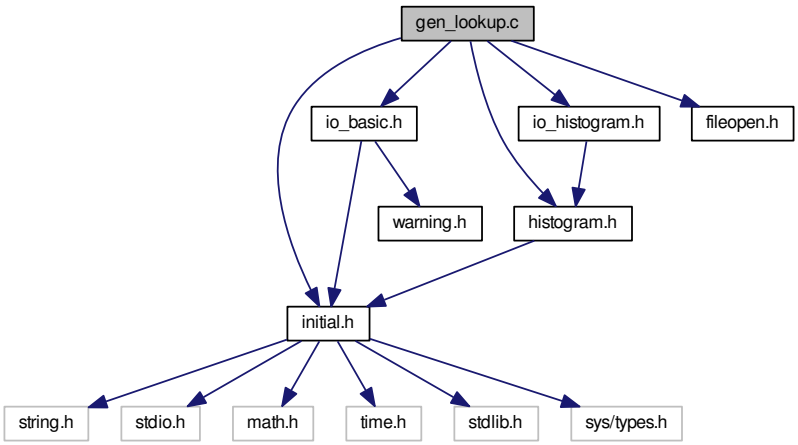
```
Filter (or filter out) some EventIO blocks in given files.
Syntax: bin/filterio [ options ] [ block types ... ] filename [ ... ]
Options:
  -s  Show statistics of input and output.
  -v  Verbose output.
  -q  Quiet mode, not even showing total blocks count.
  -n  Not accepting any blocks of the listed types.
      The default is to accept only the listed types.
  -o fname Output goes to given file instead of standard output.


@author Konrad Bernloehr
```

9.22 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**

9.22.1 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).

Date

```
CVS $Revision: 1.21 $
```

Version

```
CVS $Date: 2012/05/11 13:18:48 $
```

9.22.2 Function Documentation

9.22.2.1 void fill_gaps ( )

Fill gaps in those histograms used for generating the lookups.

Depending on the physical quantities we have different strategies for interpolation/extrapolation/smoothing.

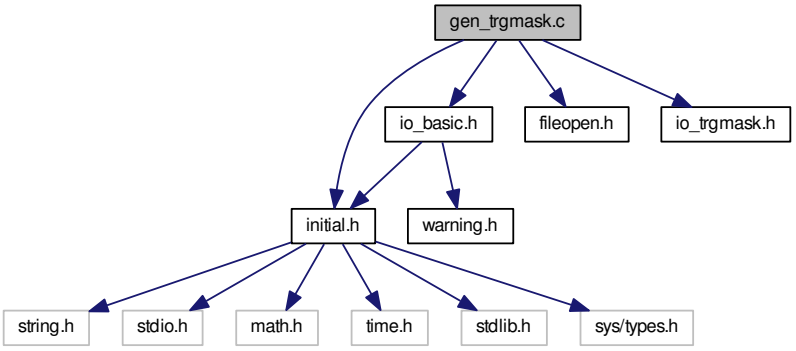
References Histogram_Extension::ddata, histogram::extension, fill_histogram(), Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, and Histogram_Parameters::upper_limit.

9.23 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)

9.23.1 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.

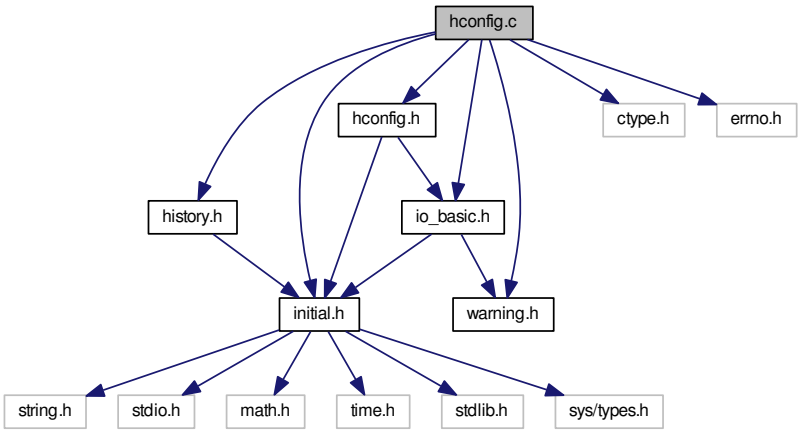
```
Syntax: bin/gen_trgmask log-file [ trgmask-file ]
    or: bin/gen_trgmask -l  trgmask-file
The first variant will create a file with a single data block
for the trigger mask patterns recovered from the log file.
The default file name is derived with extension .trgmask.gz
Note that only data for one run per file is supported.
The second variant will list the contents of such a file.


@author Konrad Bernloehr
```

9.24 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 "hconfig.h"
#include "history.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_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 f_show_config (const char *name, CONFIG_VALUES *val)

  Display the current configuration status (internal usage only).
- 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_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)
- 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)
- 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**

9.24.1 Detailed Description

Configuration control and procedure call interface.

Author

Konrad Bernloehr

Date

Date:

2014/02/20 11:40:42

Version

Revision:

1.17

```
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"),
             "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.

9.24.2 Function Documentation

9.24.2.1 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)

9.24.2.2 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.

References abbrev(), and ConfigItemStruct::name.

Referenced by f_show_config(), and reconfig().

9.24.2.3 char* get_config_filename (void)

Return the current value of the configuration file name.

Parameters

–	(none)

Returns

pointer to static file name string

9.24.2.4 char* get_config_preprocessor (void)

Return the current value of the configuration preprocessor.

Parameters

–	(none)

Returns

pointer to static command string

9.24.2.5 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)

References abbrev(), ConfigItemStruct::data, ConfigValues::data_changed, ConfigValues::data_saved, do_config(), ConfigValues::elem_size, Config_Binary_Item_Interface::elem_size, ConfigIntern::elem_size, ConfigValues::elements, find_config_binary_interface(), ConfigItemStruct::flags, ConfigItemStruct::initial, ConfigItemStruct::internal, Config_Binary_Item_Interface::io_item_type, ConfigValues::itype, ConfigIntern::itype, ConfigItemStruct::lbound, ConfigValues::list_mod, ConfigValues::max_mod, ConfigValues::mod_flag, ConfigValues::name, ConfigItemStruct::name, Config_Binary_Item_Interface::new_func, reconfig(), ConfigValues::section, ConfigItemStruct::size, ConfigItemStruct::type, ConfigItemStruct::ubound, and ConfigIntern::values.

9.24.2.6 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.

9.24.2.7 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).

9.24.2.8 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)

References do_config(), find_config_item(), getword(), ConfigItemStruct::internal, ConfigIntern::locked, and ConfigItemStruct::name.

Referenced by init_config(), and reload_config().

9.24.2.9 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)

References reconfig().

9.24.2.10 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)

9.24.2.11 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

9.24.2.12 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)

9.24.2.13 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)

9.24.3 Variable Documentation

9.24.3.1 struct config_specific_data config_defaults [static]

Initial value:

```
=
{
   "_internal_"
}
```

9.24.3.2 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) },
   { "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.

9.24.3.3 CONFIG_BLOCK first_config_block [static]

Initial value:

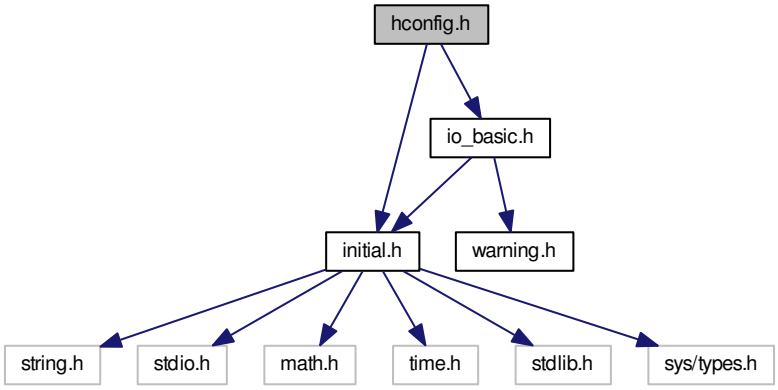
```
=
   { "_internal_", default_config, (CONFIG_BLOCK *) NULL, 0 }
```

9.25 hconfig.h File Reference

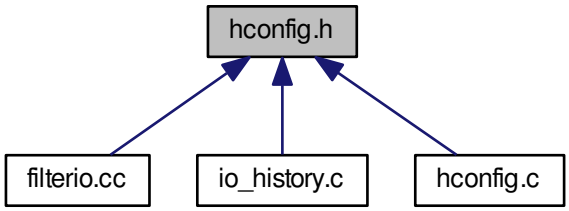
Declare hconfig structures and functions.

```
#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
- #define CFG_MUTEX(mutex) (NULL)

  Mutexes are only inserted when pthreads are used.

Typedefs

- 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.

- 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)
- 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.

9.25.1 Detailed Description

Declare hconfig structures and functions.

Author

Konrad Bernloehr

Date

CVS

Date:

2014/02/20 11:40:42

Version

CVS

Revision:

1.7

9.25.2 Macro Definition Documentation

9.25.2.1 #define _STR_(*s*) #s

Enclose in string without macro expansion.

9.25.2.2 #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.

9.25.3 Function Documentation

9.25.3.1 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.

Referenced by do_config(), find_config_item(), and init_config().

9.25.3.2 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)

9.25.3.3 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.

9.25.3.4 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.

References get_item_begin(), get_item_end(), get_string(), next_subitem_type(), _struct_IO_ITEM_HEADER::type, ev_reg_entry::type, and _struct_IO_ITEM_HEADER::version.

Referenced by config_binary_read_name().

9.25.3.5 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.

References get_item_begin(), get_short(), next_subitem_type(), _struct_IO_ITEM_HEADER::type, ev_reg_entry::type, unget_item(), and _struct_IO_ITEM_HEADER::version.

9.25.3.6 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().

9.25.3.7 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().

9.25.3.8 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.

References abbrev(), and ConfigItemStruct::name.

Referenced by f_show_config(), and reconfig().

9.25.3.9 char* get_config_filename (void)

Return the current value of the configuration file name.

Parameters

–	(none)

Returns

pointer to static file name string

9.25.3.10 char* get_config_preprocessor (void)

Return the current value of the configuration preprocessor.

Parameters

–	(none)

Returns

pointer to static command string

9.25.3.11 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 addpath(), do_config(), initpath(), main(), prog_path(), reconfig(), and user_set_tel_type_param_by_str().

9.25.3.12 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)

References abbrev(), ConfigItemStruct::data, ConfigValues::data_changed, ConfigValues::data_saved, do_config(), ConfigValues::elem_size, Config_Binary_Item_Interface::elem_size, ConfigIntern::elem_size, ConfigValues::elements, find_config_binary_interface(), ConfigItemStruct::flags, ConfigItemStruct::initial, ConfigItemStruct::internal, Config_Binary_Item_Interface::io_item_type, ConfigValues::itype, ConfigIntern::itype, ConfigItemStruct::lbound, ConfigValues::list_mod, ConfigValues::max_mod, ConfigValues::mod_flag, ConfigValues::name, ConfigItemStruct::name, Config_Binary_Item_Interface::new_func, reconfig(), ConfigValues::section, ConfigItemStruct::size, ConfigItemStruct::type, ConfigItemStruct::ubound, and ConfigIntern::values.

9.25.3.13 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.

9.25.3.14 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).

9.25.3.15 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)

References do_config(), find_config_item(), getword(), ConfigItemStruct::internal, ConfigIntern::locked, and ConfigItemStruct::name.

Referenced by init_config(), and reload_config().

9.25.3.16 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)

References reconfig().

9.25.3.17 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)

9.25.3.18 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

9.25.3.19 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)

9.25.3.20 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)

9.26 hessio_doc.h File Reference

Add an introduction to doxygen-generated documentation.

9.26.1 Detailed Description

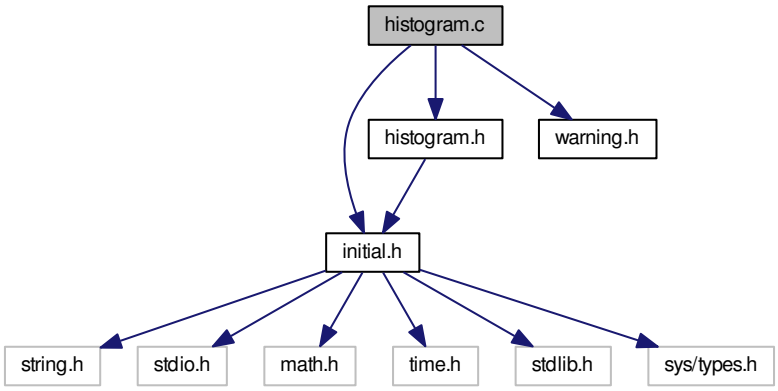
Add an introduction to doxygen-generated documentation. This file is not included during compilation.

9.27 histogram.c File Reference

Manage, fill, and display one- and two-dimensional histograms.

```
#include "initial.h"
#include "histogram.h"
#include "warning.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** (HISTOGRAM *histo)
- void **histogram_unlock** (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 (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

9.27.1 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 - 2010
CVS

Date:

2014/02/20 10:53:06

Version

CVS

Revision:

1.21

9.27.2 Macro Definition Documentation

9.27.2.1 #define HistOutput(*a*)

Value:

```
do { if ( histogram_file == (FILE *) NULL ) \
        Output(a); \
     else \
        fputs(a,histogram_file); } while(zero)
```

9.27.3 Function Documentation

9.27.3.1 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.

References Histogram_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, histogram::extension, Histogram_Extension::fdata, histogram_matching(), histogram::ident, Histogram_Parameters::integer, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and histogram::underflow_2d.

Referenced by read_histograms_x().

9.27.3.2 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(), initialize_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by allocate_histogram(), book_int_histogram(), and read_histograms_x().

9.27.3.3 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().

Referenced by read_histograms_x().

9.27.3.4 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(), initialize_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by allocate_histogram(), book_int_histogram(), and read_histograms_x().

9.27.3.5 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().

Referenced by read_histograms_x().

9.27.3.6 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

References alloc_2d_int_histogram(), alloc_int_histogram(), aux_alloc_histogram(), initialize_histogram(), Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, and Histogram_Parameters::upper_limit.

Referenced by alloc_2d_real_histogram(), alloc_real_histogram(), book_1d_histogram(), book_histogram(), and read_histograms_x().

9.27.3.7 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 mc_event_fill(), and user_init().

9.27.3.8 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().

Referenced by main(), mc_event_fill(), and user_init().

9.27.3.9 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

References alloc_2d_int_histogram(), alloc_int_histogram(), and describe_histogram().

9.27.3.10 void clear_histogram (HISTOGRAM * *histo*)

Initialize an existing histogram.

Parameters

histo	– pointer to histogram

Returns

(none)

References Histogram_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, Histogram_Parameters::integer, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and histogram::underflow_2d.

Referenced by gen_image_lookups(), histogram_to_lookup(), initialize_histogram(), and write_dst_histos().

9.27.3.11 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

References get_histogram_by_ident(), histogram::ident, and histogram::title.

Referenced by book_1d_histogram(), book_histogram(), book_int_histogram(), and read_histograms_x().

9.27.3.12 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::entries, histogram::extension, Histogram_Extension::fdata, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, histogram::type, histogram::underflow, histogram::underflow_2d, and Histogram_Parameters::upper_limit.

Referenced by display_histogram().

9.27.3.13 void display_all_histograms (void)

Display all histograms in list of histograms.

Arguments: none

Return value: none

References display_histogram(), and histogram::next.

Referenced by main().

9.27.3.14 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, display_2d_histogram(), 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.

Referenced by display_all_histograms(), and main().

9.27.3.15 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::entries, histogram::extension, Histogram_Parameters::integer, histogram::nbins_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, and histogram::type.

9.27.3.16 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 histogram::counts, histogram::entries, fill_2d_real_histogram(), fill_int_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, histogram::underflow_2d, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by fill_histogram().

9.27.3.17 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 histogram::counts, histogram::entries, fill_2d_weighted_histogram(), fill_real_histogram(), Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, histogram::underflow_2d, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_int_histogram(), and fill_histogram().

9.27.3.18 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_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::entries, histogram::extension, Histogram_Extension::fdata, fill_weighted_histogram(), histogram::ident, Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, histogram::underflow_2d, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_real_histogram(), and fill_histogram().

9.27.3.19 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 fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_int_histogram(), fill_real_histogram(), fill_weighted_histogram(), histogram::ident, histogram::nbins_2d, and histogram::type.

Referenced by fill_gaps(), fill_histogram_by_ident(), gen_image_lookups(), main(), mc_event_fill(), and user_init().

9.27.3.20 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)

References fill_histogram(), and get_histogram_by_ident().

Referenced by main(), user_event_fill(), and user_mc_event_fill().

9.27.3.21 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 histogram::counts, histogram::entries, fill_real_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::overflow, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by fill_2d_int_histogram(), and fill_histogram().

9.27.3.22 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 histogram::counts, histogram::entries, fill_weighted_histogram(), Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::overflow, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_real_histogram(), fill_histogram(), and fill_int_histogram().

9.27.3.23 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_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, Histogram_Extension::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, histogram::ident, Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::overflow, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_weighted_histogram(), fill_histogram(), and fill_real_histogram().

9.27.3.24 void free_all_histograms (void)

Deletes all histograms which are included in the linked list of histograms.

Returns

(none)

References free_histogram(), and histogram::next.

9.27.3.25 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.

Referenced by free_histogram().

9.27.3.26 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)

References free_histo_contents(), and unlink_histogram().

Referenced by free_all_histograms(), main(), read_histograms_x(), and user_init().

9.27.3.27 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(), main(), write_all_histograms(), and write_histograms().

9.27.3.28 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

References histogram::ident, and histogram::next.

Referenced by describe_histogram(), fill_histogram_by_ident(), histogram_to_root(), img_norm(), main(), read_histograms_x(), user_init(), and write_dst_histos().

9.27.3.29 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)

References histogram::ident, and histogram::next.

Referenced by mc_event_fill(), and user_init().

9.27.3.30 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.

Referenced by add_histogram().

9.27.3.31 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

References clear_histogram(), histogram::counts, histogram::entries, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::tentries, histogram::type, and histogram::underflow.

9.27.3.32 void list_histograms (long *ident*)

List all available histograms using the 'Output()' function.

Parameters

ident	– histogram ident to search or 0

Returns

(none)

References histogram::entries, histogram::ident, histogram::nbins, histogram::nbins_2d, histogram::next, histogram::tentries, histogram::title, and histogram::type.

9.27.3.33 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.

References histogram::counts, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, Histogram_Parameters::real, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

Referenced by stat_histogram().

9.27.3.34 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.

9.27.3.35 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.

9.27.3.36 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::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, 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.

Referenced by main().

9.27.3.37 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

References histogram::next, and histogram::previous.

9.27.3.38 void sort_histograms (void)

Sort histograms in linked list by idents.

Returns

(none)

References histogram::next, and histogram::previous.

Referenced by main().

9.27.3.39 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

References Histogram_Extension::content_all, Histogram_Extension::content_inside, histogram::counts, Histogram_Extension::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, Histogram_Parameters::integer, locate_histogram_fraction(), Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, and Histogram_Parameters::upper_limit.

9.27.3.40 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)

References histogram::ident, histogram::next, and histogram::previous.

Referenced by free_histogram().

9.27.4 Variable Documentation

9.27.4.1 CONST_QUAL short primetab[] [static]

Initial value:

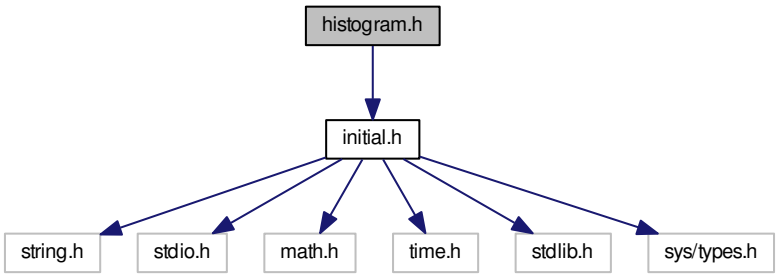
```
=
    { 131, 233, 353, 541, 751, 1051, 1367, 1511, 1723,
      1931, 2393, 3163, 3907, 5261, 6143, 7187, 8623, 9749, 11321, 15031 }
```

9.28 histogram.h File Reference

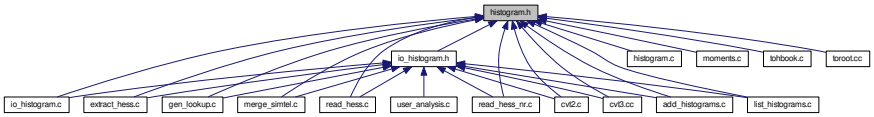
Declarations for handling one- and two-dimensional histograms.

```
#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 (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.

9.28.1 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 - 2010
CVS

Date:

2013/10/21 12:53:31

Version

CVS

Revision:

1.12

9.28.2 Typedef Documentation

9.28.2.1 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.

9.28.2.2 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).

9.28.3 Function Documentation

9.28.3.1 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.

References Histogram_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, histogram::extension, Histogram_Extension::fdata, histogram_matching(), histogram::ident, Histogram_Parameters::integer, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and histogram::underflow_2d.

Referenced by read_histograms_x().

9.28.3.2 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(), initialize_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by allocate_histogram(), book_int_histogram(), and read_histograms_x().

9.28.3.3 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().

Referenced by read_histograms_x().

9.28.3.4 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(), initialize_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by allocate_histogram(), book_int_histogram(), and read_histograms_x().

9.28.3.5 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().

Referenced by user_init().

9.28.3.6 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().

Referenced by read_histograms_x().

9.28.3.7 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

References alloc_2d_int_histogram(), alloc_int_histogram(), aux_alloc_histogram(), initialize_histogram(), Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, and Histogram_Parameters::upper_limit.

Referenced by alloc_2d_real_histogram(), alloc_real_histogram(), book_1d_histogram(), book_histogram(), and read_histograms_x().

9.28.3.8 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 mc_event_fill(), and user_init().

9.28.3.9 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().

Referenced by main(), mc_event_fill(), and user_init().

9.28.3.10 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

References alloc_2d_int_histogram(), alloc_int_histogram(), and describe_histogram().

9.28.3.11 void clear_histogram (HISTOGRAM * *histo*)

Initialize an existing histogram.

Parameters

histo	– pointer to histogram

Returns

(none)

References Histogram_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, Histogram_Parameters::integer, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and histogram::underflow_2d.

Referenced by gen_image_lookups(), histogram_to_lookup(), initialize_histogram(), and write_dst_histos().

9.28.3.12 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(), and user_event_fill().

9.28.3.13 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

References get_histogram_by_ident(), histogram::ident, and histogram::title.

Referenced by book_1d_histogram(), book_histogram(), book_int_histogram(), and read_histograms_x().

9.28.3.14 void display_all_histograms (void)

Display all histograms in list of histograms.

Arguments: none

Return value: none

References display_histogram(), and histogram::next.

Referenced by main().

9.28.3.15 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, display_2d_histogram(), 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.

Referenced by display_all_histograms(), and main().

9.28.3.16 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::entries, histogram::extension, Histogram_Parameters::integer, histogram::nbins_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, and histogram::type.

9.28.3.17 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 histogram::counts, histogram::entries, fill_2d_real_histogram(), fill_int_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, histogram::underflow_2d, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by fill_histogram().

9.28.3.18 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 histogram::counts, histogram::entries, fill_2d_weighted_histogram(), fill_real_histogram(), Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, histogram::underflow_2d, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_int_histogram(), and fill_histogram().

9.28.3.19 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_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::entries, histogram::extension, Histogram_Extension::fdata, fill_weighted_histogram(), histogram::ident, Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, histogram::underflow_2d, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_real_histogram(), and fill_histogram().

9.28.3.20 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 fill_2d_int_histogram(), fill_2d_real_histogram(), fill_2d_weighted_histogram(), fill_int_histogram(), fill_real_histogram(), fill_weighted_histogram(), histogram::ident, histogram::nbins_2d, and histogram::type.

Referenced by fill_gaps(), fill_histogram_by_ident(), gen_image_lookups(), main(), mc_event_fill(), and user_init().

9.28.3.21 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)

References fill_histogram(), and get_histogram_by_ident().

Referenced by main(), user_event_fill(), and user_mc_event_fill().

9.28.3.22 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 histogram::counts, histogram::entries, fill_real_histogram(), Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::overflow, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, Histogram_Parameters::upper_limit, and Histogram_Parameters::width.

Referenced by fill_2d_int_histogram(), and fill_histogram().

9.28.3.23 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

9.28.3.24 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

9.28.3.25 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

Referenced by user_event_fill().

9.28.3.26 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 histogram::counts, histogram::entries, fill_weighted_histogram(), Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::overflow, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_real_histogram(), fill_histogram(), and fill_int_histogram().

9.28.3.27 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

9.28.3.28 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

9.28.3.29 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

9.28.3.30 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_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, Histogram_Extension::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, histogram::ident, Histogram_Parameters::inverse_binwidth, Histogram_Parameters::lower_limit, histogram::nbins, histogram::overflow, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

Referenced by fill_2d_weighted_histogram(), fill_histogram(), and fill_real_histogram().

9.28.3.31 void free_all_histograms (void)

Deletes all histograms which are included in the linked list of histograms.

Returns

(none)

References free_histogram(), and histogram::next.

9.28.3.32 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)

References free_histo_contents(), and unlink_histogram().

Referenced by free_all_histograms(), main(), read_histograms_x(), and user_init().

9.28.3.33 void free_moments (MOMENTS * *mom*)

Deallocates memory previously allocated to a moments structure.

Parameters

mom	Pointer to previously allocated structure

9.28.3.34 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(), main(), write_all_histograms(), and write_histograms().

9.28.3.35 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

References histogram::ident, and histogram::next.

Referenced by describe_histogram(), fill_histogram_by_ident(), histogram_to_root(), img_norm(), main(), read_histograms_x(), user_init(), and write_dst_histos().

9.28.3.36 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)

References histogram::ident, and histogram::next.

Referenced by mc_event_fill(), and user_init().

9.28.3.37 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.

Referenced by add_histogram().

9.28.3.38 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

References clear_histogram(), histogram::counts, histogram::entries, histogram::nbins, histogram::nbins_2d, histogram::overflow, histogram::tentries, histogram::type, and histogram::underflow.

9.28.3.39 void list_histograms (long *ident*)

List all available histograms using the 'Output()' function.

Parameters

ident	– histogram ident to search or 0

Returns

(none)

References histogram::entries, histogram::ident, histogram::nbins, histogram::nbins_2d, histogram::next, histogram::tentries, histogram::title, and histogram::type.

9.28.3.40 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.

References histogram::counts, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::overflow, Histogram_Parameters::real, histogram::type, histogram::underflow, and Histogram_Parameters::upper_limit.

Referenced by stat_histogram().

9.28.3.41 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.

9.28.3.42 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.

9.28.3.43 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::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, 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.

Referenced by main().

9.28.3.44 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

References histogram::next, and histogram::previous.

9.28.3.45 void sort_histograms (void)

Sort histograms in linked list by idents.

Returns

(none)

References histogram::next, and histogram::previous.

Referenced by main().

9.28.3.46 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

References Histogram_Extension::content_all, Histogram_Extension::content_inside, histogram::counts, Histogram_Extension::ddata, histogram::entries, histogram::extension, Histogram_Extension::fdata, Histogram_Parameters::integer, locate_histogram_fraction(), Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, histogram::type, and Histogram_Parameters::upper_limit.

9.28.3.47 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)

Referenced by user_event_fill().

9.28.3.48 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)

References histogram::ident, histogram::next, and histogram::previous.

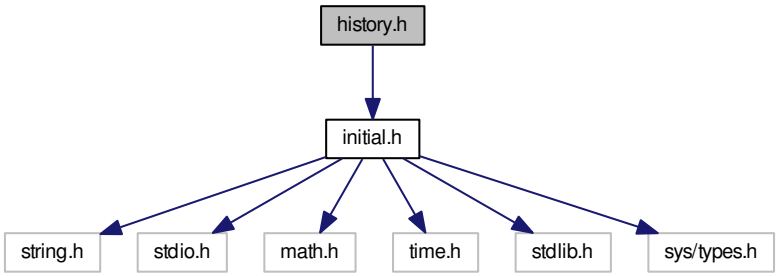
Referenced by free_histogram().

9.29 history.h File Reference

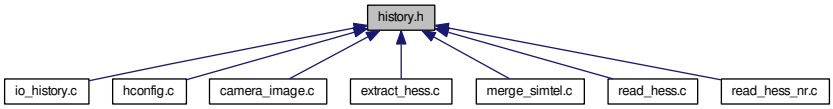
Keep blocks of history in the data (like command line of programs operating on the data, ...)

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

Include dependency graph for history.h:



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



Functions

- int **push_command_history** (int argc, char **argv)
- int **push_config_history** (const char *line, int replace)
- int **write_history** (long id, IO_BUFFER *iobuf)
- int **write_config_history** (const char *htext, long htime, long id, IO_BUFFER *iobuf)
- int **list_history** (IO_BUFFER *iobuf, FILE *file)

9.29.1 Detailed Description

Keep blocks of history in the data (like command line of programs operating on the data, ...)

Author

Konrad Bernloehr

Date

1997 to 2010

```
$Date: 2014/02/20 11:40:42 $
```

Version

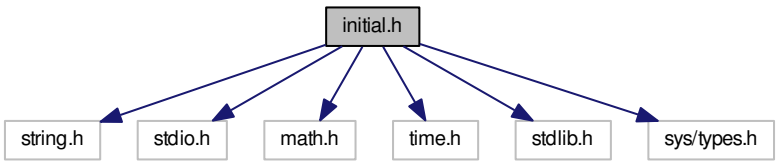
```
$Revision: 1.5 $
```

9.30 initial.h File Reference

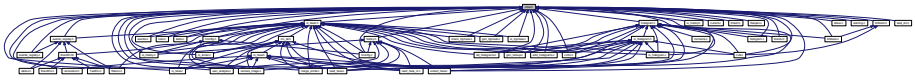
Indentification of the system and including some basic include file.

```
#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**

9.30.1 Detailed Description

Indentification of the system and including some basic include file.

```
@author  Konrad Bernloehr
@date    1991 to 2010
@date    @verbatim $Date: 2012/11/13 16:28:15 $
```

Version

```
$Revision: 1.14 $
```

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,
```

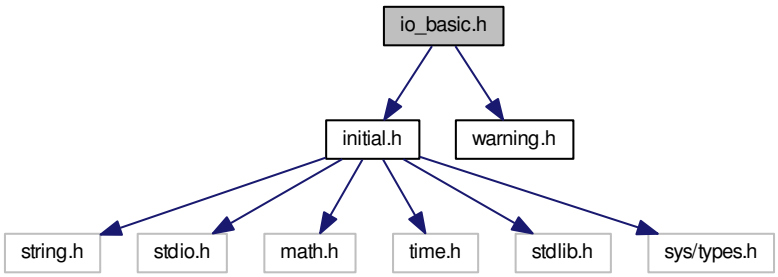
```
    OS_DARWIN (Mac OS X).
    Note: ULTRIX may be on VAX or MIPS, LINUX on Intel or Alpha,
    OS_LYNX on 68K or PowerPC.
OS_OS9

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
```

9.31 io_basic.h File Reference

Basic header file for eventio data format.

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

Include dependency graph for io_basic.h:



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

Data Structures

- struct _struct_IO_ITEM_HEADER

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

  The IO_BUFFER structure contains all data needed the manage the stuff.
- struct ev_reg_entry

Macros

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

Typedefs

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

Functions

- IO_BUFFER * allocate_io_buffer (size_t buflen)

  Dynamic allocation of an I/O buffer.
- int extend_io_buffer (IO_BUFFER *iobuf, unsigned next_byte, long increment)

  Extend the dynamically allocated I/O buffer.
- void free_io_buffer (IO_BUFFER *iobuf)

  Free an I/O buffer that has been allocated at run-time.
- void put_vector_of_byte (const BYTE *vec, int num, IO_BUFFER *iobuf)

  Put a vector of bytes into an I/O buffer.
- void get_vector_of_byte (BYTE *vec, int num, IO_BUFFER *iobuf)

  Get a vector of bytes from an I/O buffer.
- void put_count (uintmax_t num, IO_BUFFER *iobuf)

  Put an unsigned integer of unspecified length to an I/O buffer.
- void put_count32 (uint32_t num, IO_BUFFER *iobuf)

  Shortened version of put_count for up to 32 bits of data.
- void put_count16 (uint16_t num, IO_BUFFER *iobuf)

  Shortened version of put_count for up to 16 bits of data.
- uintmax_t get_count (IO_BUFFER *iobuf)

  Get an unsigned integer of unspecified length from an I/O buffer.
- uint32_t get_count32 (IO_BUFFER *iobuf)

  Get an unsigned 32 bit integer of unspecified length from an I/O buffer.
- uint16_t get_count16 (IO_BUFFER *iobuf)

  Get an unsigned 16 bit integer of unspecified length from an I/O buffer.

- void put_scount (intmax_t num, IO_BUFFER *iobuf)

  Put a signed integer of unspecified length to an I/O buffer.
- void put_scount32 (int32_t num, IO_BUFFER *iobuf)

  Shorter version of put_scount for up to 32 bytes of data.
- void put_scount16 (int16_t num, IO_BUFFER *iobuf)

  Shorter version of put_scount for up to 16 bytes of data.
- intmax_t get_scount (IO_BUFFER *iobuf)

  Get a signed integer of unspecified length from an I/O buffer.
- int32_t get_scount32 (IO_BUFFER *iobuf)

  Shortened version of get_scount for up to 32 bits of data.
- int16_t get_scount16 (IO_BUFFER *iobuf)

  Shortened version of get_scount for up to 16 bits of data.
- void put_vector_of_int_scount (const int *vec, int num, IO_BUFFER *iobuf)

  Put an array of ints as scount32 data into an I/O buffer.
- void get_vector_of_int_scount (int *vec, int num, IO_BUFFER *iobuf)

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

  Put a vector of 2-byte integers on an I/O buffer.
- void get_vector_of_short (short *vec, int num, IO_BUFFER *iobuf)

  Get a vector of short integers from I/O buffer.
- void put_vector_of_uint16 (const uint16_t *uval, int num, IO_BUFFER *iobuf)

  Put a vector of unsigned shorts into an I/O buffer.
- void get_vector_of_uint16 (uint16_t *uval, int num, IO_BUFFER *iobuf)

  Get a vector of unsigned shorts from an I/O buffer.
- uint16_t get_uint16 (IO_BUFFER *iobuf)

  Get one unsigned short from an I/O buffer.
- void put_short (int num, IO_BUFFER *iobuf)

  Put a two-byte integer on an I/O buffer.
- int get_short (IO_BUFFER *iobuf)

  Get a two-byte integer from an I/O buffer.
- void put_vector_of_int (const int *vec, int num, IO_BUFFER *iobuf)

  Put a vector of integers (range -32768 to 32767) into I/O buffer.
- void get_vector_of_int (int *vec, int num, IO_BUFFER *iobuf)

  Get a vector of (small) integers from I/O buffer.
- void put_int32 (int32_t num, IO_BUFFER *iobuf)

  Write a four-byte integer to an I/O buffer.
- int32_t get_int32 (IO_BUFFER *iobuf)

  Read a four byte integer from an I/O buffer.
- void put_uint32 (uint32_t num, IO_BUFFER *iobuf)

  Put a four-byte integer into an I/O buffer.
- uint32_t get_uint32 (IO_BUFFER *iobuf)

  Get a four-byte unsigned integer from an I/O buffer.
- void put_vector_of_int32 (const int32_t *vec, int num, IO_BUFFER *iobuf)

  Put a vector of 32 bit integers into I/O buffer.
- void get_vector_of_int32 (int32_t *vec, int num, IO_BUFFER *iobuf)

  Get a vector of 32 bit integers from I/O buffer.
- void put_vector_of_uint32 (const uint32_t *vec, int num, IO_BUFFER *iobuf)

  Put a vector of 32 bit integers into I/O buffer.
- void get_vector_of_uint32 (uint32_t *vec, int num, IO_BUFFER *iobuf)

  Get a vector of 32 bit integers from I/O buffer.
- void put_long (long num, IO_BUFFER *iobuf)

Put a four-byte integer taken from a 'long' into an I/O buffer.

- long get_long (IO_BUFFER *iobuf)

  Get 4-byte integer from I/O buffer and return as a long int.
- void put_vector_of_long (const long *vec, int num, IO_BUFFER *iobuf)

  Put a vector of long int as 4-byte integers into an I/O buffer.
- void get_vector_of_long (long *vec, int num, IO_BUFFER *iobuf)

  Get a vector of 4-byte integers as long int from I/O buffer.
- int put_string (const char *s, IO_BUFFER *iobuf)

  Put a string of ASCII characters into an I/O buffer.
- int get_string (char *s, int nmax, IO_BUFFER *iobuf)

  Get a string of ASCII characters from an I/O buffer.
- int put_long_string (const char *s, IO_BUFFER *iobuf)

  Put a long string of ASCII characters into an I/O buffer.
- int get_long_string (char *s, int nmax, IO_BUFFER *iobuf)

  Get a long string of ASCII characters from an I/O buffer.
- int put_var_string (const char *s, IO_BUFFER *iobuf)

  Put a string of ASCII characters into an I/O buffer.
- int get_var_string (char *s, int nmax, IO_BUFFER *iobuf)

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

  Put a vector of floats as IEEE 'float' numbers into an I/O buffer.
- void get_vector_of_float (float *vec, int num, IO_BUFFER *iobuf)

  Get a vector of floating point numbers as 'floats' from an I/O buffer.
- void put_real (double d, IO_BUFFER *iobuf)

  Put a 4-byte floating point number into an I/O buffer.
- double get_real (IO_BUFFER *iobuf)

  Get a floating point number (as written by put_real) from the I/O buffer.
- void put_vector_of_real (const double *vec, int num, IO_BUFFER *iobuf)

  Put a vector of doubles as IEEE 'float' numbers into an I/O buffer.
- void get_vector_of_real (double *vec, int num, IO_BUFFER *iobuf)

  Get a vector of floating point numbers as 'doubles' from an I/O buffer.
- void **put_double** (double d, IO_BUFFER *iobuf)
- double **get_double** (IO_BUFFER *iobuf)
- void **put_vector_of_double** (const double *vec, int num, IO_BUFFER *iobuf)
- void **get_vector_of_double** (double *vec, int num, IO_BUFFER *iobuf)
- void dbl_to_sfloat (double dnum, uint16_t *snum)

  Convert a double to the internal representation of a 16 bit floating point number as specified in the OpenGL 3.1 standard.
- void put_sfloat (double dnum, IO_BUFFER *iobuf)

  Put a 16-bit float to an I/O buffer.
- double dbl_from_sfloat (const uint16_t *snum)

  Convert from the internal representation of an OpenGL 16-bit floating point number back to normal floating point representation.
- double get_sfloat (IO_BUFFER *iobuf)

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

  Begin putting another (sub-) item into the output buffer.
- int put_item_begin_with_flags (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, int user_flag, int extended)

  Begin putting another (sub-) item into the output buffer.
- int put_item_end (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

End of putting an item into the output buffer.

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

  Undo writing at the present level.
- int get_item_begin (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Begin reading an item.
- int get_item_end (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  End reading an item.
- int unget_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Go back to the beginning of an item being read.
- int next_subitem_type (IO_BUFFER *iobuf)

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

  Reads the header of a sub-item and return the length of it.
- long next_subitem_ident (IO_BUFFER *iobuf)

  Reads the header of a sub-item and return the identifier of it.
- int skip_subitem (IO_BUFFER *iobuf)

  When the next sub-item is of no interest, it can be skipped.
- int search_sub_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, IO_ITEM_HEADER *sub_item_header)

  Search for an item of a specified type.
- int rewind_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Go back to the beginning of an item.
- int remove_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Remove an item from an I/O buffer.
- int list_sub_items (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, int maxlevel, int verbosity)

  Display the contents of sub-items on standard output.
- int reset_io_block (IO_BUFFER *iobuf)

  Reset an I/O block to its empty status.
- int write_io_block (IO_BUFFER *iobuf)

  Write an I/O block to the block's output.
- int find_io_block (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Find the beginning of the next I/O data block in the input.
- int read_io_block (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Read the data of an I/O block from the input.
- int skip_io_block (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header)

  Skip the data of an I/O block from the input.
- int list_io_blocks (IO_BUFFER *iobuf, int verbosity)

  Show the top-level item of an I/O block on standard output.
- int copy_item_to_io_block (IO_BUFFER *iobuf2, IO_BUFFER *iobuf, const IO_ITEM_HEADER *item_header)

  Copy a sub-item to another I/O buffer as top-level item.
- int append_io_block_as_item (IO_BUFFER *iobuf, IO_ITEM_HEADER *item_header, BYTE *_buffer, long length)

  Append data from one I/O block into another one.
- struct ev_reg_entry * find_ev_reg (unsigned long t)

  This optionally available function is implemented externally.
- void set_eventio_registry_hook (EVREGSEARCH fptr)

  This function should be used to set the find_ev_reg_ptr function pointer.
- const char * eventio_registered_typename (unsigned long type)

  This functions using the stored function pointer are now in the core eventio code.
- const char * eventio_registered_description (unsigned long type)

  Extract the optional description for a given type number, if available.

9.31.1 Detailed Description

Basic header file for eventio data format.

Author

Konrad Bernloehr

Date

1991 to 2014

```
CVS $Date: 2014/06/03 16:19:44 $
```

Version

```
CVS $Revision: 1.21 $
```

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

9.31.2 Macro Definition Documentation

9.31.2.1 #define put_byte(_c, _p)

Value:

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

9.31.3 Function Documentation

9.31.3.1 IO_BUFFER* allocate_io_buffer (size_t *buflen*)

Dynamic allocation of an I/O buffer.

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

Parameters

buflen	The length of the actual buffer in bytes. A safety margin of 4 bytes is added.

Returns

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

References _struct_IO_BUFFER::aux_count, _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, _struct_IO_BUFFER::extended, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::is_allocated, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_BUFFER::max_length, _struct_IO_BUFFER::min_length, _struct_IO_BUFFER::output_file, _struct_IO_BUFFER::output_fileno, _struct_IO_BUFFER::regular, _struct_IO_BUFFER::sub_item_length, _struct_IO_BUFFER::sync_err_count, _struct_IO_BUFFER::sync_err_max, _struct_IO_BUFFER::user_function, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::EventIO(), main(), and write_all_histograms().

9.31.3.2 int append_io_block_as_item (IO_BUFFER * *iobuf,* IO_ITEM_HEADER * *item_header,* BYTE * *buffer,* long *length*)

Append data from one I/O block into another one.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, extend_io_buffer(), _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::sub_item_length, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Append().

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

Copy a sub-item to another I/O buffer as top-level item.

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, reset_io_block(), _struct_IO_BUFFER::sub_item_length, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Copy(), and main().

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

Convert a double to the internal representation of a 16 bit floating point number as specified in the OpenGL 3.1 standard.

Parameters

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

Referenced by put_sfloat().

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

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

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

References find_ev_reg(), ev_reg_entry::name, and none.

Referenced by list_io_blocks(), list_sub_items(), main(), and eventio::EventIO::Item::TypeName().

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

Extend the dynamically allocated I/O buffer.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::is_allocated, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::max_length, and _struct_IO_BUFFER::w_remaining.

Referenced by append_io_block_as_item(), copy_item_to_io_block(), put_int32(), put_item_begin_with_flags(), put_long(), put_short(), put_uint32(), put_vector_of_byte(), put_vector_of_uint16(), and read_io_block().

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

Find the beginning of the next I/O data block in the input.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, get_item_begin(), get_long(), get_uint32(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::sync_err_count, _struct_IO_BUFFER::sync_err_max, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_BUFFER::user_function, _struct_IO_ITEM_HEADER::version, and _struct_IO_BUFFER::w_remaining.

Referenced by check_autoload_trgmask(), eventio::EventIO::Find(), list_io_blocks(), and main().

9.31.3.8 void free_io_buffer (IO_BUFFER * *iobuf*)

Free an I/O buffer that has been allocated at run-time.

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

Parameters

iobuf	The buffer descriptor to be de-allocated.

Returns

(none)

References _struct_IO_BUFFER::buffer, and _struct_IO_BUFFER::is_allocated.

Referenced by write_all_histograms(), and eventio::EventIO::~EventIO().

9.31.3.9 uintmax_t get_count (IO_BUFFER * *iobuf*)

Get an unsigned integer of unspecified length from an I/O buffer.

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

Referenced by get_scount(), get_var_string(), print_trgmask(), read_test1(), read_test2(), read_test3(), and read_trgmask().

9.31.3.10 uint16_t get_count16 (IO_BUFFER * *iobuf*)

Get an unsigned 16 bit integer of unspecified length from an I/O buffer.

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

Referenced by get_scount16(), eventio::EventIO::Item::GetCount16(), and read_test1().

9.31.3.11 uint32_t get_count32 (IO_BUFFER * *iobuf*)

Get an unsigned 32 bit integer of unspecified length from an I/O buffer.

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

Referenced by get_scount32(), print_hess_centralevent(), read_hess_centralevent(), and read_test1().

9.31.3.12 int32_t get_int32 (IO_BUFFER * *iobuf*)

Read a four byte integer from an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_inquire_numbers(), config_binary_read_numbers(), get_long_string(), get_real(), eventio::EventIO::Item::GetInt32(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixelset(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_televt_head(), read_hess_camorgan(),

read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_televt_head(), and read_test1().

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

Begin reading an item.

Reads the header of an item.

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

Parameters

iobuf	The input buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::byte_order, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, get_long(), get_uint32(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::length, _struct_IO_ITEM_HEADER::level, _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, _struct_IO_ITEM_HEADER::version, and _struct_IO_BUFFER::w_remaining.

Referenced by begin_read_tel_array(), config_binary_inquire_numbers(), config_binary_read_index(), config_binary_read_numbers(), config_binary_read_text(), config_binary_text_length(), find_io_block(), eventio::EventIO::Item::Item(), list_sub_items(), main(), next_subitem_ident(), next_subitem_length(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pixtime(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_array_end(), read_tel_array_head(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), read_test3(), read_trgmask(), search_sub_item(), and skip_subitem().

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

End reading an item.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::w_remaining.

Referenced by config_binary_read_index(), config_binary_read_numbers(), config_binary_read_text(), eventio::EventIO::Item::Done(), end_read_tel_array(), list_sub_items(), main(), print_camera_layout(), print_hess_calib_event(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_event(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_phot(), print_hess_mc_run_stat(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixel_list(), print_hess_pixelset(), print_hess_pixtime(), print_hess_run_stat(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televent(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), print_trgmask(), read_camera_layout(), read_hess_calib_event(), read_hess_camorgan(), read_hess_camsettings(), read_hess_camsoftset(), read_hess_centralevent(), read_hess_event(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_pe_sum(), read_hess_mc_phot(), read_hess_mc_run_stat(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixel_list(), read_hess_pixeldis(), read_hess_pixelset(), read_hess_pixtime(), read_hess_pointingcor(), read_hess_run_stat(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televent(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_array_end(), read_tel_array_head(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), read_test3(), read_trgmask(), search_sub_item(), and skip_subitem().

9.31.3.15 long get_long (IO_BUFFER * *iobuf*)

Get 4-byte integer from I/O buffer and return as a long int.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_read_index(), find_io_block(), get_item_begin(), get_real(), get_time_blob(), next_subitem_type(), print_hess_runheader(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), read_hess_runheader(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_histograms_x(), read_input_lines(), read_photo_electrons(), read_shower_longitudinal(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), read_test1(), read_test2(), and read_test3().

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

Get a long string of ASCII characters from an I/O buffer.

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

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

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

References _struct_IO_BUFFER::data, get_int32(), and get_vector_of_byte().

Referenced by read_test1(), read_test2(), and read_test3().

9.31.3.17 double get_real (IO_BUFFER * *iobuf*)

Get a floating point number (as written by put_real) from the I/O buffer.

Parameters

iobuf	The I/O buffer descriptor;

Returns

The floating point number.

References get_int32(), and get_long().

Referenced by get_vector_of_float(), get_vector_of_real(), eventio::EventIO::Item::GetReal(), print_camera_layout(), print_hess_camorgan(), print_hess_camsettings(), print_hess_centralevent(), print_hess_laser_calib(), print_hess_mc_event(), print_hess_mc_pe_sum(), print_hess_mc_shower(), print_hess_mcrunheader(), print_hess_pixelset(), print_hess_pixtime(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_televt_head(), print_hess_telimage(), print_hess_trackevent(), print_histograms(), print_photo_electrons(), print_tel_block(), print_tel_offset(), print_tel_photons(), print_tel_pos(), read_hess_camorgan(), read_hess_camsettings(), read_hess_centralevent(), read_hess_laser_calib(), read_hess_mc_event(), read_hess_mc_shower(), read_hess_mcrunheader(), read_hess_pixelset(), read_hess_pixtime(), read_hess_runheader(), read_hess_shower(), read_hess_tel_monitor(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackevent(), read_hess_trackset(), read_histograms_x(), read_photo_electrons(), read_shower_longitudinal(), read_tel_block(), read_tel_offset_w(), read_tel_photons(), read_tel_pos(), and read_test1().

9.31.3.18 intmax_t get_scount (IO_BUFFER * *iobuf*)

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

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

References get_count().

Referenced by print_hess_camorgan(), print_hess_pixelset(), print_hess_teladc_samples(), read_hess_camorgan(), read_hess_pixelset(), read_hess_teladc_samples(), read_test1(), read_test2(), and read_test3().

9.31.3.19 int get_short (IO_BUFFER * *iobuf*)

Get a two-byte integer from an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_text_length(), get_string(), get_vector_of_int(), get_vector_of_short(), eventio::EventIO::Item::GetInt16(), eventio::EventIO::Item::GetUint16(), print_camera_layout(), print_hess_camorgan(), print_hess_centralevent(), print_hess_laser_calib(), print_hess_mc_pe_sum(), print_hess_mc_shower(), print_hess_pixel_list(), print_hess_pixtime(), print_hess_runheader(), print_hess_shower(), print_hess_tel_monitor(), print_hess_teladc_samples(), print_hess_teladc_sums(), print_hess_televt_head(), print_hess_telimage(), print_histograms(), print_photo_electrons(), print_tel_photons(), read_camera_layout(), read_hess_camorgan(), read_hess_centralevent(), read_hess_laser_calib(), read_hess_mc_pe_sum(), read_hess_mc_shower(), read_hess_pixel_list(), read_hess_pixtime(), read_hess_shower(), read_hess_tel_monitor(), read_hess_teladc_samples(), read_hess_teladc_sums(), read_hess_televt_head(), read_hess_telimage(), read_hess_trackset(), read_histograms_x(), read_photo_electrons(), read_shower_longitudinal(), read_tel_photons(), and read_test1().

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

Get a string of ASCII characters from an I/O buffer.

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

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

References _struct_IO_BUFFER::data, get_short(), and get_vector_of_byte().

Referenced by config_binary_read_text(), print_hess_runheader(), print_histograms(), read_hess_runheader(), read_histograms_x(), read_input_lines(), read_test1(), read_test2(), and read_test3().

9.31.3.21 uint16_t get_uint16 (IO_BUFFER * iobuf)

Get one unsigned short from an I/O buffer.

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

Parameters

iobuf	The output buffer descriptor.

Returns

The value obtained from the I/O buffer.

References get_vector_of_uint16().

Referenced by get_sfloat(), and print_hess_tel_monitor().

9.31.3.22 uint32_t get_uint32 (IO_BUFFER * iobuf)

Get a four-byte unsigned integer from an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by find_io_block(), get_item_begin(), eventio::EventIO::Item::GetUint32(), and read_test1().

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

Get a string of ASCII characters from an I/O buffer.

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

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

References _struct_IO_BUFFER::data, get_count(), and get_vector_of_byte().

Referenced by eventio::EventIO::Item::GetString(), read_test1(), read_test2(), and read_test3().

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

Get a vector of bytes from an I/O buffer.

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::data.

Referenced by config_binary_read_numbers(), get_long_string(), get_string(), get_var_string(), read_hess_tel_monitor(), read_test2(), and read_test3().

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

Get a vector of unsigned shorts from an I/O buffer.

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

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::byte_order, and _struct_IO_BUFFER::data.

Referenced by config_binary_read_numbers(), get_uint16(), print_hess_teladc_samples(), read_hess_tel_monitor(), read_hess_teladc_samples(), and read_hess_teladc_sums().

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

Show the top-level item of an I/O block on standard output.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::byte_order, eventio_registered_description(), eventio_registered_typename(), find_io_block(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, skip_io_block(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::user_flag, and _struct_IO_ITEM_HEADER::version.

Referenced by eventio::EventIO::List(), and main().

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

Display the contents of sub-items on standard output.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, eventio_registered_description(), eventio_registered_typename(), get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, list_sub_items(), search_sub_item(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, and _struct_IO_ITEM_HEADER::version.

Referenced by eventio::EventIO::Item::List(), list_sub_items(), and main().

9.31.3.28 long next_subitem_ident (IO_BUFFER * *iobuf*)

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

Parameters

iobuf	The input buffer descriptor.

Returns

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

References _struct_IO_BUFFER::data, get_item_begin(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and unget_item().

Referenced by eventio::EventIO::Item::NextSubItemIdent(), and read_hess_televent().

9.31.3.29 long next_subitem_length (IO_BUFFER * *iobuf*)

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

Parameters

iobuf	The input buffer descriptor.

Returns

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

References _struct_IO_BUFFER::data, get_item_begin(), _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_ITEM_HEADER::type, and unget_item().

Referenced by eventio::EventIO::Item::NextSubItemLength().

9.31.3.30 int next_subitem_type (IO_BUFFER * *iobuf*)

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

Parameters

iobuf	The input buffer descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, get_long(), _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, and _struct_IO_BUFFER::item_start_offset.

Referenced by config_binary_read_text(), config_binary_text_length(), eventio::EventIO::Item::NextSubItemType(), print_hess_event(), print_hess_mc_phot(), print_hess_televent(), read_hess_event(), read_hess_mc_phot(), read_hess_televent(), and read_test3().

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

Put an unsigned integer of unspecified length to an I/O buffer.

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

Parameters

n	The number to be saved. Even on systems with 64-bit integers, this must not exceed 2**32-1 with the current implementation.
iobuf	The output buffer descriptor.

Returns

(none)

References put_vector_of_byte().

Referenced by put_scount(), put_scount16(), put_scount32(), put_var_string(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Shortened version of put_count for up to 16 bits of data.

Returns

(none)

References put_vector_of_byte().

Referenced by eventio::EventIO::Item::PutCount16(), write_test1(), write_test2(), and write_test3().

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

Shortened version of put_count for up to 32 bits of data.

Returns

(none)

References put_vector_of_byte().

Referenced by write_hess_centralevent(), write_test1(), write_test2(), and write_test3().

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

Write a four-byte integer to an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_long_string(), put_real(), put_vector_of_int32(), eventio::EventIO::Item::PutInt32(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_televt_head(), and write_test1().

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

Begin putting another (sub-) item into the output buffer.

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

Parameters

iobuf	The output buffer descriptor.
item_header	The item header descriptor.

Returns

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

References put_item_begin_with_flags().

Referenced by begin_write_tel_array(), config_binary_envelope_begin(), config_binary_write_index(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Begin putting another (sub-) item into the output buffer.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, extend_io_buffer(), _struct_IO_BUFFER::extended, _struct_IO_ITEM_HEADER::ident, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::length, _struct_IO_ITEM_HEADER::level, put_long(), _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_ITEM_HEADER::user_flag, _struct_IO_ITEM_HEADER::version, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Item::Item(), and put_item_begin().

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

End of putting an item into the output buffer.

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

Parameters

iobuf	The output buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::length, _struct_IO_ITEM_HEADER::level, put_uint32(), _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::use_extension, _struct_IO_BUFFER::w_remaining, and write_io_block().

Referenced by config_binary_envelope_end(), config_binary_write_index(), eventio::EventIO::Item::Done(), end_write_tel_array(), write_camera_layout(), write_hess_calib_event(), write_hess_camorgan(), write_hess_camsettings(), write_hess_camsoftset(), write_hess_centralevent(), write_hess_event(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_pe_sum(), write_hess_mc_run_stat(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixel_list(), write_hess_pixeldis(), write_hess_pixelset(), write_hess_pixtime(), write_hess_pointingcor(), write_hess_run_stat(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televent(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_array_end(), write_tel_array_head(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Put a four-byte integer taken from a 'long' into an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by config_binary_write_index(), put_item_begin_with_flags(), put_real(), put_time_blob(), put_vector_of_long(), write_hess_runheader(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_histograms(), write_input_lines(), write_photo_electrons(), write_shower_longitudinal(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), write_tel_pos(), write_test1(), write_test2(), and write_test3().

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

Put a long string of ASCII characters into an I/O buffer.

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

Parameters

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

Returns

Length of string

References put_int32(), put_short(), and put_vector_of_byte().

Referenced by write_test1(), write_test2(), and write_test3().

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

Put a 4-byte floating point number into an I/O buffer.

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

Parameters

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

Returns

(none)

References put_int32(), and put_long().

Referenced by put_vector_of_float(), put_vector_of_real(), eventio::EventIO::Item::PutReal(), write_hess_camorgan(), write_hess_camsettings(), write_hess_centralevent(), write_hess_laser_calib(), write_hess_mc_event(), write_hess_mc_shower(), write_hess_mcrunheader(), write_hess_pixelset(), write_hess_pixtime(), write_hess_runheader(), write_hess_shower(), write_hess_tel_monitor(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackevent(), write_hess_trackset(), write_histograms(), write_shower_longitudinal(), write_tel_block(), write_tel_compact_photons(), write_tel_offset_w(), write_tel_photons(), and write_test1().

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

Put a signed integer of unspecified length to an I/O buffer.

Put a signed integer of unspecified length in a way similar to the UTF-8 character encoding to an I/O buffer. The byte order resulting in the buffer is independent of the host byte order or the byte order in action for the I/O buffer, starting

with as many leading bits in the first byte as extension bytes needed after the first byte. While the scheme in principle allows for values of arbitrary length, the implementation is limited to 32 bits. To allow an efficient representation of negative numbers, the sign bit is stored in the least significant bit. Portability of data across machines with different intmax_t sizes and the need to represent also the most negative number (-(2^31), -(2^63), or -(2^127), depending on CPU type and compiler) is achieved by putting the number's modulus minus 1 into the higher bits.

Parameters

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

Returns

(none)

References put_count().

Referenced by write_hess_pixel_list(), write_hess_pixelset(), write_hess_teladc_samples(), write_test1(), write_test2(), and write_test3().

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

Shorter version of put_scount for up to 16 bytes of data.

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

Returns

(none)

References put_count().

Referenced by eventio::EventIO::Item::PutSCount16(), write_test1(), write_test2(), and write_test3().

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

Shorter version of put_scount for up to 32 bytes of data.

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

Returns

(none)

References put_count().

Referenced by put_vector_of_int_scount(), write_hess_camorgan(), write_hess_pixtime(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_televt_head(), write_hess_telimage(), write_test1(), write_test2(), write_test3(), and write_trgmask().

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

Put a two-byte integer on an I/O buffer.

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

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_long_string(), put_string(), put_vector_of_int(), put_vector_of_short(), write_camera_layout(), write_hess_camorgan(), write_hess_centralevent(), write_hess_laser_calib(), write_hess_mc_pe_sum(), write_hess_mc_shower(), write_hess_pixel_list(), write_hess_pixtime(), write_hess_shower(), write_hess_tel_monitor(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televt_head(), write_hess_telimage(), write_hess_trackset(), write_histograms(), write_photo_electrons(), write_shower_longitudinal(), write_tel_compact_photons(), write_tel_photons(), and write_test1().

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

Put a string of ASCII characters into an I/O buffer.

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

Parameters

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

Returns

Length of string

References put_short(), and put_vector_of_byte().

Referenced by write_hess_runheader(), write_histograms(), write_input_lines(), write_test1(), write_test2(), and write_test3().

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

Put a four-byte integer into an I/O buffer.

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

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_item_end(), put_vector_of_uint32(), eventio::EventIO::Item::PutUint32(), remove_item(), and write_test1().

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

Put a string of ASCII characters into an I/O buffer.

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

Parameters

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

Returns

Length of string

References put_count(), and put_vector_of_byte().

Referenced by write_test1(), write_test2(), and write_test3().

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

Put a vector of bytes into an I/O buffer.

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_count(), put_count16(), put_count32(), put_long_string(), put_string(), put_var_string(), write_hess_tel_monitor(), write_test2(), and write_test3().

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

Put a vector of integers (range -32768 to 32767) into I/O buffer.

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

References put_short().

Referenced by write_hess_camorgan(), write_hess_centralevent(), write_hess_mc_pe_sum(), write_hess_pixel_list(), write_hess_pixelset(), write_hess_pixtime(), write_hess_runheader(), write_hess_shower(), write_hess_teladc_sums(), write_hess_telimage(), write_test2(), and write_test3().

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

Put a vector of 2-byte integers on an I/O buffer.

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

References put_short().

Referenced by write_hess_tel_monitor(), write_test2(), and write_test3().

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

Put a vector of unsigned shorts into an I/O buffer.

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

Parameters

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

Returns

(none)

References _struct_IO_BUFFER::byte_order, _struct_IO_BUFFER::data, extend_io_buffer(), and _struct_IO_BUFFER::w_remaining.

Referenced by put_sfloat(), eventio::EventIO::Item::PutUint16(), write_hess_tel_monitor(), write_hess_teladc_samples(), and write_hess_teladc_sums().

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

Read the data of an I/O block from the input.

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

Parameters

iobuf	The I/O buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data_pending, extend_io_buffer(), _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, skip_io_block(), _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::user_function.

Referenced by check_autoload_trgmask(), main(), and eventio::EventIO::Read().

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

Remove an item from an I/O buffer.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, put_uint32(), _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::use_extension, and _struct_IO_BUFFER::w_remaining.

9.31.3.54 int reset_io_block (IO_BUFFER * *iobuf*)

Reset an I/O block to its empty status.

Parameters

iobuf	The I/O buffer descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::data_pending, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::min_length, _struct_IO_BUFFER::regular, _struct_IO_BUFFER::sub_item_length, and _struct_IO_BUFFER::w_remaining.

Referenced by copy_item_to_io_block(), and main().

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

Go back to the beginning of an item.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, and _struct_IO_BUFFER::w_remaining.

Referenced by read_test3(), and eventio::EventIO::Item::Rewind().

9.31.3.56 int search_sub_item (IO_BUFFER * *iobuf,* IO_ITEM_HEADER * *item_header,* IO_ITEM_HEADER * *sub_item_header*)

Search for an item of a specified type.

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

Parameters

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

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_ITEM_HEADER::can_search, _struct_IO_BUFFER::data, get_item_begin(), get_item_end(), _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::w_remaining.

Referenced by list_sub_items(), read_test3(), and eventio::EventIO::Item::Search().

9.31.3.57 void set_eventio_registry_hook (EVREGSEARCH *fptr*)

This function should be used to set the find_ev_reg_ptr function pointer.

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

Skip the data of an I/O block from the input.

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

Parameters

iobuf	The I/O buffer descriptor.
item_header	The item header descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data_pending, _struct_IO_BUFFER::input_file, _struct_IO_BUFFER::input_fileno, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::regular, _struct_IO_BUFFER::sub_item_length, _struct_IO_ITEM_HEADER::type, and _struct_IO_BUFFER::user_function.

Referenced by list_io_blocks(), main(), read_io_block(), and eventio::EventIO::Skip().

9.31.3.59 int skip_subitem (IO_BUFFER * *iobuf*)

When the next sub-item is of no interest, it can be skipped.

Parameters

iobuf	I/O buffer descriptor.

Returns

0 (ok), -1 (error)

References get_item_begin(), get_item_end(), and _struct_IO_ITEM_HEADER::type.

Referenced by print_hess_event(), print_hess_mc_phot(), print_hess_televent(), read_hess_mc_phot(), read_hess_televent(), and eventio::EventIO::Item::Skip().

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

Go back to the beginning of an item being read.

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

Parameters

iobuf	I/O buffer descriptor.

item_header	Header of item last read.

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, and _struct_IO_BUFFER::w_remaining.

Referenced by config_binary_inquire_numbers(), config_binary_read_numbers(), config_binary_text_length(), next_subitem_ident(), next_subitem_length(), read_photo_electrons(), read_tel_photons(), and eventio::EventIO::Item::Unget().

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

Undo writing at the present level.

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

Parameters

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

Returns

0 (ok), -1 (error)

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::buflen, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::item_start_offset, _struct_IO_ITEM_HEADER::level, _struct_IO_ITEM_HEADER::use_extension, and _struct_IO_BUFFER::w_remaining.

Referenced by eventio::EventIO::Item::Unput(), write_hess_event(), and write_hess_televent().

9.31.3.62 int write_io_block (IO_BUFFER * *iobuf*)

Write an I/O block to the block's output.

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

Parameters

iobuf	The I/O buffer descriptor.

Returns

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

References _struct_IO_BUFFER::buffer, _struct_IO_BUFFER::data, _struct_IO_BUFFER::item_extension, _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_BUFFER::output_file, _struct_IO_BUFFER::output_fileno, _struct_IO_BUFFER::user_function, and _struct_IO_BUFFER::w_remaining.

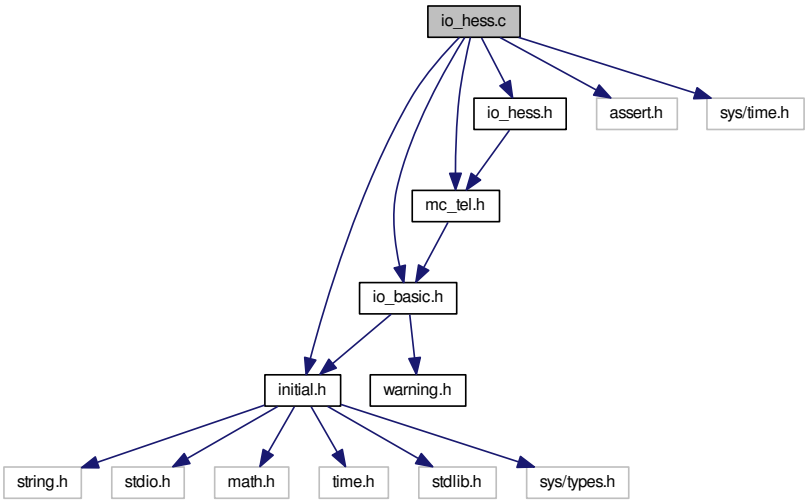
Referenced by main(), put_item_end(), eventio::EventIO::Write(), and write_io_block_to_file().

9.32 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 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** ()
- 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_hess_runheader (IO_BUFFER *iobuf, RunHeader *rh)

  Write the run header in eventio format.
- int read_hess_runheader (IO_BUFFER *iobuf, RunHeader *rh)

  Read the run header in eventio format.
- int print_hess_runheader (IO_BUFFER *iobuf)

  Read the run header in eventio format.
- int write_hess_mcrunheader (IO_BUFFER *iobuf, MCRunHeader *mcrh)

  Write the Monte Carlo run header in eventio format.
- int read_hess_mcrunheader (IO_BUFFER *iobuf, MCRunHeader *mcrh)

Read the Monte Carlo run header in eventio format.

- int print_hess_mcrunheader (IO_BUFFER *iobuf)

  Print the Monte Carlo run header data.
- int write_hess_camsettings (IO_BUFFER *iobuf, CameraSettings *cs)

  Write the camera definition (pixel positions) in eventio format.
- int read_hess_camsettings (IO_BUFFER *iobuf, CameraSettings *cs)

  Read the camera definition (pixel positions) in eventio format.
- int print_hess_camsettings (IO_BUFFER *iobuf)

  Print the camera definition (pixel positions) in eventio format.
- int write_hess_camorgan (IO_BUFFER *iobuf, CameraOrganisation *co)

  Write the logical organisation of camera electronics in eventio format.
- int read_hess_camorgan (IO_BUFFER *iobuf, CameraOrganisation *co)

  Read the logical organisation of camera electronics in eventio format.
- int print_hess_camorgan (IO_BUFFER *iobuf)

  Read the logical organisation of camera electronics in eventio format.
- int write_hess_pixelset (IO_BUFFER *iobuf, PixelSetting *ps)

  Write the settings of pixel parameters (HV, thresholds, ...) in eventio format.
- int read_hess_pixelset (IO_BUFFER *iobuf, PixelSetting *ps)

  Read the settings of pixel parameters (HV, thresholds, ...) in eventio format.
- int print_hess_pixelset (IO_BUFFER *iobuf)

  Show the settings of pixel parameters (HV, thresholds, ...) in eventio format.
- int write_hess_pixeldis (IO_BUFFER *iobuf, PixelDisabled *pd)

  Write which pixels are disabled in HV and/or trigger in eventio format.
- int read_hess_pixeldis (IO_BUFFER *iobuf, PixelDisabled *pd)

  Read which pixels are disabled in HV and/or trigger in eventio format.
- int write_hess_camsoftset (IO_BUFFER *iobuf, CameraSoftSet *cs)

  Write camera software parameters relevant for data recording in eventio format.
- int read_hess_camsoftset (IO_BUFFER *iobuf, CameraSoftSet *cs)

  Read camera software parameters relevant for data recording in eventio format.
- int write_hess_trackset (IO_BUFFER *iobuf, TrackingSetup *ts)

  Write the settings for tracking of a telescope in eventio format.
- int read_hess_trackset (IO_BUFFER *iobuf, TrackingSetup *ts)

  Read the settings for tracking of a telescope in eventio format.
- int write_hess_pointingcor (IO_BUFFER *iobuf, PointingCorrection *pc)

  Write the parameters of a telescope's pointing correction in eventio format.
- int read_hess_pointingcor (IO_BUFFER *iobuf, PointingCorrection *pc)

  Read the parameters of a telescope's pointing correction in eventio format.
- int write_hess_centralevent (IO_BUFFER *iobuf, CentralEvent *ce)

  Write the trigger data of the central trigger in eventio format.
- int read_hess_centralevent (IO_BUFFER *iobuf, CentralEvent *ce)

  Read the trigger data of the central trigger in eventio format.
- int print_hess_centralevent (IO_BUFFER *iobuf)

  Print the trigger data of the central trigger in eventio format.
- int write_hess_trackevent (IO_BUFFER *iobuf, TrackEvent *tke)

  Write a tracking position in eventio format.
- int read_hess_trackevent (IO_BUFFER *iobuf, TrackEvent *tke)

  Read a tracking position in eventio format.
- int print_hess_trackevent (IO_BUFFER *iobuf)

  Print the tracking data in eventio format.
- int write_hess_televt_head (IO_BUFFER *iobuf, TelEvent *te)

  Write the event header for data from one camera in eventio format.

- int read_hess_televt_head (IO_BUFFER *iobuf, TelEvent *te)

  Read the event header for data from one camera in eventio format.
- int print_hess_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_hess_teladc_sums (IO_BUFFER *iobuf, AdcData *raw)

  Write ADC sum data for one camera in eventio format.
- int read_hess_teladc_sums (IO_BUFFER *iobuf, AdcData *raw)

  Write ADC sum data for one camera in eventio format.
- int print_hess_teladc_sums (IO_BUFFER *iobuf)

  Print summed ADC data in eventio format.
- int write_hess_teladc_samples (IO_BUFFER *iobuf, AdcData *raw)

  Write sampled ADC data in eventio format.
- int read_hess_teladc_samples (IO_BUFFER *iobuf, AdcData *raw, int what)

  Read sampled ADC data in eventio format.
- int print_hess_teladc_samples (IO_BUFFER *iobuf)

  Print sampled ADC data in eventio format.
- static void **adc_reset** (AdcData *raw)
- 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_hess_pixtime (IO_BUFFER *iobuf, PixelTiming *pixtm)

  Write pixel timing parameters for selected pixels.
- int read_hess_pixtime (IO_BUFFER *iobuf, PixelTiming *pixtm)

  Read pixel timing parameters for selected pixels.
- int print_hess_pixtime (IO_BUFFER *iobuf)

  Print sampled ADC data in eventio format.
- int write_hess_telimage (IO_BUFFER *iobuf, ImgData *img, int what)

  Write image parameters for one telescope in eventio format.
- int read_hess_telimage (IO_BUFFER *iobuf, ImgData *img)

  Read image parameters for one telescope in eventio format.
- int print_hess_telimage (IO_BUFFER *iobuf)

  Print image parameters for one telescope in eventio format.
- int write_hess_televent (IO_BUFFER *iobuf, TelEvent *te, int what)

  Write data for one telescope camera in eventio format.
- int read_hess_televent (IO_BUFFER *iobuf, TelEvent *te, int what)

  Read data for one telescope camera in eventio format.
- int print_hess_televent (IO_BUFFER *iobuf)

  Print data for one telescope camera in eventio format.
- int write_hess_shower (IO_BUFFER *iobuf, ShowerParameters *sp)

  Write reconstructed shower parameters in eventio format.
- int read_hess_shower (IO_BUFFER *iobuf, ShowerParameters *sp)

  Read reconstructed shower parameters in eventio format.
- int print_hess_shower (IO_BUFFER *iobuf)

  Print reconstructed shower parameters in eventio format.
- int write_hess_event (IO_BUFFER *iobuf, FullEvent *ev, int what)

  Write the full array data of one event in eventio format.

- int read_hess_event (IO_BUFFER *iobuf, FullEvent *ev, int what)

  Read the full array data of one event in eventio format.
- int print_hess_event (IO_BUFFER *iobuf)

  Print the full array data of one event in eventio format.
- int write_hess_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_hess_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_hess_calib_event (IO_BUFFER *iobuf)

  Print a calibration event (pedestal, laser, led, ...) as an encapsulated raw data event.
- int write_hess_mc_shower (IO_BUFFER *iobuf, MCShower *mcs)

  Write MC data for one simulated shower in eventio format.
- int read_hess_mc_shower (IO_BUFFER *iobuf, MCShower *mcs)

  Read MC data for one simulated shower in eventio format.
- int print_hess_mc_shower (IO_BUFFER *iobuf)

  Print MC data for one simulated shower in eventio format.
- int write_hess_mc_event (IO_BUFFER *iobuf, MCEvent *mce)

  Write MC data for one use of a simulated shower in eventio format.
- int read_hess_mc_event (IO_BUFFER *iobuf, MCEvent *mce)

  Read MC data for one use of a simulated shower in eventio format.
- int print_hess_mc_event (IO_BUFFER *iobuf)

  Print MC data for one use of a simulated shower in eventio format.
- int write_hess_mc_pe_sum (IO_BUFFER *iobuf, MCpeSum *mcpes)

  Write the numbers of photo-electrons detected from Cherenkov light in eventio format.
- int read_hess_mc_pe_sum (IO_BUFFER *iobuf, MCpeSum *mcpes)

  Read the numbers of photo-electrons detected from Cherenkov light in eventio format.
- int print_hess_mc_pe_sum (IO_BUFFER *iobuf)

  Print the numbers of photo-electrons detected from Cherenkov light in eventio format.
- 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_hess_tel_monitor (IO_BUFFER *iobuf, TelMoniData *mon, int what)

  Write telescope camera monitoring information in eventio format.
- int read_hess_tel_monitor (IO_BUFFER *iobuf, TelMoniData *mon)

  Read telescope camera monitoring information in eventio format.
- int print_hess_tel_monitor (IO_BUFFER *iobuf)

  Print telescope camera monitoring information in eventio format.
- int write_hess_laser_calib (IO_BUFFER *iobuf, LasCalData *lcd)

  Write a set of laser calibration data in eventio format.
- int read_hess_laser_calib (IO_BUFFER *iobuf, LasCalData *lcd)

  Read a set of laser calibration data in eventio format.
- int print_hess_laser_calib (IO_BUFFER *iobuf)

  Print a set of laser calibration data in eventio format.
- int write_hess_run_stat (IO_BUFFER *iobuf, RunStat *rs)

  Write run statistics in eventio format.
- int read_hess_run_stat (IO_BUFFER *iobuf, RunStat *rs)

  Read run statistics in eventio format.
- int print_hess_run_stat (IO_BUFFER *iobuf)

  Print run statistics in eventio format.

- int write_hess_mc_run_stat (IO_BUFFER *iobuf, MCRunStat *mcrs)

  Write Monte Carlo run statistics in eventio format.
- int read_hess_mc_run_stat (IO_BUFFER *iobuf, MCRunStat *mcrs)

  Read Monte Carlo run statistics in eventio format.
- int print_hess_mc_run_stat (IO_BUFFER *iobuf)

  Print Monte Carlo run statistics in eventio format.
- int read_hess_mc_phot (IO_BUFFER *iobuf, MCEvent *mce)

  Read Monte Carlo photons and photo-electrons.
- int print_hess_mc_phot (IO_BUFFER *iobuf)

  Print Monte Carlo photons and photo-electrons.
- int write_hess_pixel_list (IO_BUFFER *iobuf, PixelList *pl, int telescope)

  Write lists of pixels (triggered, selected in image analysis, ...)
- int read_hess_pixel_list (IO_BUFFER *iobuf, PixelList *pl, int *telescope)

  Read lists of pixels (triggered, selected in image analysis, ...)
- int print_hess_pixel_list (IO_BUFFER *iobuf)

  Print lists of pixels (triggered, selected in image analysis, ...)

Variables

- static int **g_tel_idx** [3][H_MAX_TEL+1]
- static int **g_tel_idx_init** [3]
- static int **g_tel_idx_ref**

9.32.1 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)

```
CVS $Date: 2014/06/25 12:54:07 $
```

Version

```
CVS $Revision: 1.84 $
```

9.32.2 Function Documentation

9.32.2.1 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.

References H_MAX_GAINS, H_MAX_HOTPIX, H_MAX_PROFILE, H_MAX_SLICES, and H_MAX_TEL.

9.32.2.2 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).

Referenced by main(), print_hess_event(), read_hess_event(), and which_telescope_type().

9.32.2.3 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, ...

Referenced by read_hess_runheader(), and write_hess_runheader().

9.32.2.4 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().

9.32.2.5 int write_hess_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).

References hess_event_data_struct::central, hess_tracking_event_data_struct::cor_known, hess_central_event_data_struct::cpu_time, hess_central_event_data_struct::glob_count, hess_central_event_data_struct::gps_time, _struct_IO_ITEM_HEADER::ident, hess_tel_event_data_struct::loc_count, hess_event_data_struct::num_tel, hess_central_event_data_struct::num_teldata, hess_central_event_data_struct::num_teltrg, put_item_begin(), put_item_end(), hess_tracking_event_data_struct::raw_known, RAWDATA_FLAG, hess_event_data_struct::shower, hess_tel_event_data_struct::tel_id, hess_event_data_struct::teldata, hess_central_event_data_struct::teldata_list, hess_central_event_data_struct::teltrg_list, hess_central_event_data_struct::teltrg_time, hess_central_event_data_struct::teltrg_type_mask, hess_event_data_struct::trackdata, _struct_IO_ITEM_HEADER::type, unput_item(), _struct_IO_ITEM_HEADER::version, write_hess_centralevent(), write_hess_shower(), write_hess_televent(), and write_hess_trackevent().

Referenced by main(), and write_hess_calib_event().

9.32.2.6 int write_hess_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).

References hess_laser_calib_data_struct::calib, _struct_IO_ITEM_HEADER::ident, hess_laser_calib_data_struct::lascal_id, hess_laser_calib_data_struct::max_int_frac, hess_laser_calib_data_struct::max_pixtm_frac, hess_laser_calib_data_struct::num_gains, hess_laser_calib_data_struct::num_pixels, put_int32(), put_item_begin(), put_item_end(), put_real(), put_short(), put_vector_of_real(), hess_laser_calib_data_struct::tel_id, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by merge_data_from_io_block().

9.32.2.7 int write_hess_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.

References hess_mc_event_struct::aweight, hess_mc_event_struct::event, _struct_IO_ITEM_HEADER::ident, put_int32(), put_item_begin(), put_item_end(), put_real(), hess_mc_event_struct::shower_num, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, hess_mc_event_struct::xcore, and hess_mc_event_struct::ycore.

Referenced by main().

9.32.2.8 int write_hess_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.

References hess_mc_pe_sum_struct::event, _struct_IO_ITEM_HEADER::ident, hess_mc_pe_sum_struct::num_pe, hess_mc_pe_sum_struct::num_pixels, hess_mc_pe_sum_struct::num_tel, hess_mc_pe_sum_struct::photons, hess_mc_pe_sum_struct::photons_atm, hess_mc_pe_sum_struct::photons_atm_3_6, hess_mc_pe_sum_struct::photons_atm_400, hess_mc_pe_sum_struct::photons_atm_qe, hess_mc_pe_sum_struct::pix_pe, put_int32(), put_item_begin(), put_item_end(), put_short(), put_vector_of_int(), put_vector_of_int32(), put_vector_of_real(), hess_mc_pe_sum_struct::shower_num, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.32.2.9 int write_hess_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.

References hess_mc_shower_struct::altitude, hess_mc_shower_struct::azimuth, hess_mc_shower_struct::cmax, hess_mc_shower_profile_struct::content, hess_mc_shower_struct::depth_start, hess_mc_shower_struct::emax, hess_mc_shower_profile_struct::end, hess_mc_shower_struct::energy, hess_mc_shower_struct::h_first_int, hess_mc_shower_struct::hmax, hess_mc_shower_profile_struct::id, _struct_IO_ITEM_HEADER::ident, shower_extra_parameters::is_set, hess_mc_shower_struct::num_profiles, hess_mc_shower_profile_struct::num_steps, hess_mc_shower_struct::primary_id, put_int32(), put_item_begin(), put_item_end(), put_real(), put_short(), put_vector_of_real(), hess_mc_shower_profile_struct::start, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, and hess_mc_shower_struct::xmax.

Referenced by main().

9.32.2.10 int write_hess_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.

References _struct_IO_ITEM_HEADER::ident, hess_run_end_statistics_struct::num_central_trig, hess_run_end_statistics_struct::num_events, hess_run_end_statistics_struct::num_local_sys_trig, hess_run_end_statistics_struct::num_local_trig, hess_run_end_statistics_struct::num_tel, put_int32(), put_item_begin(), put_item_end(), put_vector_of_int32(), hess_run_end_statistics_struct::run_num, hess_run_end_statistics_struct::tel_ids, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.32.2.11 int write_hess_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').

References hess_shower_parameter::Alt, hess_shower_parameter::Az, hess_shower_parameter::energy, hess_shower_parameter::err_core1, hess_shower_parameter::err_core2, hess_shower_parameter::err_core3, hess_shower_parameter::err_dir1, hess_shower_parameter::err_dir2, hess_shower_parameter::err_dir3, _struct_IO_ITEM_HEADER::ident, hess_shower_parameter::img_list, hess_shower_parameter::img_pattern, hess_shower_parameter::mscl, hess_shower_parameter::mscw, hess_shower_parameter::num_img, hess_shower_parameter::num_read, hess_shower_parameter::num_trg, put_int32(), put_item_begin(), put_item_end(), put_real(), put_short(), put_vector_of_int(), hess_shower_parameter::result_bits, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, hess_shower_parameter::xc, hess_shower_parameter::xmax, and hess_shower_parameter::yc.

Referenced by write_hess_event().

9.32.2.12 int write_hess_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 hess_tel_monitor_struct::camera_temp, hess_tel_monitor_struct::coinc_count, copy_htime(), hess_tel_monitor_struct::current, hess_tel_monitor_struct::daq_conf, hess_tel_monitor_struct::data_rate, hess_tel_monitor_struct::dc_rate_time, hess_tel_monitor_struct::drawer_temp, hess_tel_monitor_struct::event_count, hess_tel_monitor_struct::event_rate, fill_htime_now(), hess_tel_monitor_struct::hv_dac, hess_tel_monitor_struct::hv_i_mon, hess_tel_monitor_struct::hv_set, hess_tel_monitor_struct::hv_stat, hess_tel_monitor_struct::hv_temp_time, hess_tel_monitor_struct::hv_v_mon, _struct_IO_ITEM_HEADER::ident, hess_tel_monitor_struct::known, hess_tel_monitor_struct::mean_significant, hess_tel_monitor_struct::moni_time, hess_tel_monitor_struct::monitor_id, hess_tel_monitor_struct::new_parts, hess_tel_monitor_struct::noise, hess_tel_monitor_struct::num_camera_temp, hess_tel_monitor_struct::num_drawer_temp, hess_tel_monitor_struct::num_drawers, hess_tel_monitor_struct::num_ped_slices, hess_tel_monitor_struct::num_pixels, hess_tel_monitor_struct::num_sectors, hess_tel_monitor_struct::ped_noise_time, hess_tel_monitor_struct::pedestal, put_int32(), put_item_begin(), put_item_end(), put_long(), put_real(), put_scount32(), put_short(), put_time_blob(), put_vector_of_byte(), put_vector_of_real(), put_vector_of_short(), put_vector_of_uint16(), hess_tel_monitor_struct::scaler, hess_tel_monitor_struct::sector_rate, hess_tel_monitor_struct::set_daq_time, hess_tel_monitor_struct::set_hv_thr_time, hess_tel_monitor_struct::status_bits, hess_tel_monitor_struct::tel_id, hess_tel_monitor_struct::thresh_dac, hess_tel_monitor_struct::trig_set, hess_tel_monitor_struct::trig_time, hess_tel_monitor_struct::trigger_rate, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by merge_data_from_io_block().

9.32.2.13 int write_hess_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 hess_tel_event_adc_struct::adc_sample, H_MAX_GAINS, _struct_IO_ITEM_HEADER::ident, hess_tel_event_adc_struct::known, hess_tel_event_adc_struct::num_gains, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::num_samples, put_item_begin(), put_item_end(), put_long(), put_scount(), put_scount32(), put_short(), put_vector_of_uint16(), hess_tel_event_adc_struct::significant, hess_tel_event_adc_struct::tel_id, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by write_hess_televent().

9.32.2.14 int write_hess_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 hess_tel_event_adc_struct::adc_list, hess_tel_event_adc_struct::adc_sum, hess_tel_event_adc_struct::data_red_mode, H_MAX_GAINS, HI_GAIN, _struct_IO_ITEM_HEADER::ident, hess_tel_event_adc_struct::known, hess_tel_event_adc_struct::list_known, hess_tel_event_adc_struct::list_size, LO_GAIN, hess_tel_event_adc_struct::num_gains, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::offset_hg8, put_item_begin(), put_item_end(), put_long(), put_short(), put_vector_of_int(), put_vector_of_uint16(), hess_tel_event_adc_struct::scale_hg8, hess_tel_event_adc_struct::significant, hess_tel_event_adc_struct::tel_id, hess_tel_event_adc_struct::threshold, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by write_hess_televent().

9.32.2.15 int write_hess_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.

References hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sum, hess_tel_event_data_struct::glob_count, H_MAX_SLICES, _struct_IO_ITEM_HEADER::ident, hess_tel_event_data_struct::image_pixels, hess_tel_event_data_struct::img, hess_tel_event_adc_struct::known, hess_pixel_timing_struct::known, hess_tel_image_struct::known, hess_tel_event_adc_struct::num_gains, hess_tel_event_data_struct::num_image_sets, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_pixel_list::pixels, hess_tel_event_data_struct::pixtm, put_item_begin(), put_item_end(), hess_tel_event_data_struct::raw, RAWDATA_FLAG, hess_tel_event_data_struct::readout_mode, hess_tel_event_adc_struct::significant, hess_tel_event_data_struct::tel_id, hess_pixel_timing_struct::timval, hess_tel_event_data_struct::trigger_pixels, _struct_IO_ITEM_HEADER::type, unput_item(), _struct_IO_ITEM_HEADER::version, write_hess_pixel_list(), write_hess_pixtime(), write_hess_teladc_samples(), write_hess_teladc_sums(), write_hess_televt_head(), write_hess_telimage(), and hess_tel_event_adc_struct::zero_sup_mode.

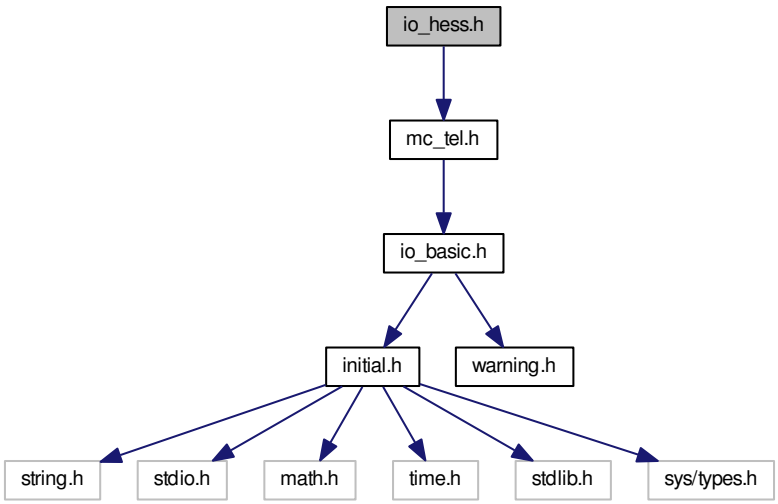
Referenced by write_hess_event().

9.33 io_hess.h File Reference

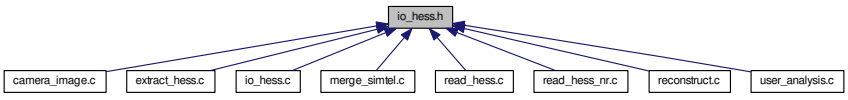
Definition and structures for H.E.S.S.

```
#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 hess_run_header_struct

  Run header common to measured and simulated data.
- struct hess_mc_run_header_struct

  MC run header.
- struct hess_camera_settings_struct

  Definition of camera optics settings.
- struct hess_camera_organisation_struct

  Logical organisation of camera electronics channels.
- struct hess_pixel_setting_struct

  Settings of pixel HV and thresholds.
- struct hess_pixel_disabled_struct

  Pixels disabled in HV and/or trigger.
- struct hess_camera_software_setting_struct

  Software settings used in camera process.
- struct hess_tracking_setup_struct

  Definition of tracking parameters.
- struct hess_pointing_correction_struct

Pointing correction parameters.

- struct hess_time_struct

  Breakdown of time into seconds since 1970.0 and nanoseconds.
- struct hess_tel_event_adc_struct

  ADC data (either sampled or sum mode)
- struct hess_pixel_timing_struct
- struct hess_pixel_list

  Lists of pixels (triggered, selected, etc.)
- struct hess_tel_image_struct

  Image parameters.
- struct hess_tel_event_data_struct

  Event raw and image data from one telescope.
- struct hess_central_event_data_struct

  Central trigger event data.
- struct hess_tracking_event_data_struct

  Tracking data interpolated for one event and one telescope.
- struct hess_shower_parameter

  Reconstructed shower parameters.
- struct hess_event_data_struct

  All data for one event.
- struct hess_mc_shower_profile_struct

  Monte Carlo shower profile (sort of histogram).
- struct hess_mc_shower_struct

  Shower specific data.
- struct hess_mc_pe_sum_struct

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

  Photons from Monte Carlo.
- struct hess_mc_pe_list

  Photo-electrons from Monte Carlo individually.
- struct hess_mc_event_struct

  Monte Carlo event-specific data.
- struct hess_tel_monitor_struct

  Monitoring data.
- struct hess_laser_calib_data_struct

  Laser calibration data.
- struct hess_run_end_statistics_struct

  End-of-run statistics.
- struct hess_run_end_mc_statistics_struct

  MC end-of-run statistics.
- struct hess_all_data_struct

  Container for all H.E.S.S.

Macros

- #define **IO_HESS_VERSION** 2
- #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 1000

  Max.
- #define H_CHECK_MAX()

  Macro expanding into a function call checking if user function.
- #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 IO_TYPE_HESS_BASE 2000

  Never change the following numbers after MC data is created:
- #define **IO_TYPE_HESS_RUNHEADER** (IO_TYPE_HESS_BASE+0)
- #define **IO_TYPE_HESS_MCRUNHEADER** (IO_TYPE_HESS_BASE+1)
- #define **IO_TYPE_HESS_CAMSETTINGS** (IO_TYPE_HESS_BASE+2)
- #define **IO_TYPE_HESS_CAMORGAN** (IO_TYPE_HESS_BASE+3)
- #define **IO_TYPE_HESS_PIXELSET** (IO_TYPE_HESS_BASE+4)
- #define **IO_TYPE_HESS_PIXELDISABLE** (IO_TYPE_HESS_BASE+5)
- #define **IO_TYPE_HESS_CAMSOFTSET** (IO_TYPE_HESS_BASE+6)
- #define **IO_TYPE_HESS_POINTINGCOR** (IO_TYPE_HESS_BASE+7)
- #define **IO_TYPE_HESS_TRACKSET** (IO_TYPE_HESS_BASE+8)
- #define **IO_TYPE_HESS_CENTEVENT** (IO_TYPE_HESS_BASE+9)
- #define **IO_TYPE_HESS_TRACKEVENT** (IO_TYPE_HESS_BASE+100)
- #define **IO_TYPE_HESS_TELEVENT** (IO_TYPE_HESS_BASE+200)
- #define **IO_TYPE_HESS_EVENT** (IO_TYPE_HESS_BASE+10)

- #define **IO_TYPE_HESS_TELEVTHEAD** (IO_TYPE_HESS_BASE+11)
- #define **IO_TYPE_HESS_TELADCSUM** (IO_TYPE_HESS_BASE+12)
- #define **IO_TYPE_HESS_TELADCSAMP** (IO_TYPE_HESS_BASE+13)
- #define **IO_TYPE_HESS_TELIMAGE** (IO_TYPE_HESS_BASE+14)
- #define **IO_TYPE_HESS_SHOWER** (IO_TYPE_HESS_BASE+15)
- #define **IO_TYPE_HESS_PIXELTIMING** (IO_TYPE_HESS_BASE+16)
- #define **IO_TYPE_HESS_MC_SHOWER** (IO_TYPE_HESS_BASE+20)
- #define **IO_TYPE_HESS_MC_EVENT** (IO_TYPE_HESS_BASE+21)
- #define **IO_TYPE_HESS_TEL_MONI** (IO_TYPE_HESS_BASE+22)
- #define **IO_TYPE_HESS_LASCAL** (IO_TYPE_HESS_BASE+23)
- #define **IO_TYPE_HESS_RUNSTAT** (IO_TYPE_HESS_BASE+24)
- #define **IO_TYPE_HESS_MC_RUNSTAT** (IO_TYPE_HESS_BASE+25)
- #define **IO_TYPE_HESS_MC_PE_SUM** (IO_TYPE_HESS_BASE+26)
- #define **IO_TYPE_HESS_PIXELLIST** (IO_TYPE_HESS_BASE+27)
- #define **IO_TYPE_HESS_CALIBEVENT** (IO_TYPE_HESS_BASE+28)
- #define **HAS_CORSIKA_INTERACTION_DETAIL** 1
- #define H_MAX_PIX_TIMES 7

  In addition to ADC we may (optionally) also have 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).

Typedefs

- typedef struct hess_run_header_struct **RunHeader**
- typedef struct hess_mc_run_header_struct **MCRunHeader**
- typedef struct hess_camera_settings_struct **CameraSettings**
- typedef struct hess_camera_organisation_struct **CameraOrganisation**
- typedef struct hess_pixel_setting_struct **PixelSetting**
- typedef struct hess_pixel_disabled_struct **PixelDisabled**
- typedef struct hess_camera_software_setting_struct **CameraSoftSet**
- typedef struct hess_tracking_setup_struct **TrackingSetup**
- typedef struct hess_pointing_correction_struct **PointingCorrection**
- typedef struct hess_time_struct **HTime**
- typedef struct hess_tel_event_adc_struct **AdcData**
- typedef struct hess_pixel_timing_struct **PixelTiming**

- typedef struct hess_pixel_list **PixelList**
- typedef struct hess_tel_image_struct **ImgData**
- typedef struct hess_tel_event_data_struct **TelEvent**
- typedef struct hess_central_event_data_struct **CentralEvent**
- typedef struct hess_tracking_event_data_struct **TrackEvent**
- typedef struct hess_shower_parameter **ShowerParameters**
- typedef struct hess_event_data_struct **FullEvent**
- typedef struct hess_mc_shower_profile_struct **ShowerProfile**
- typedef struct hess_mc_shower_struct **MCShower**
- typedef struct hess_mc_pe_sum_struct **MCpeSum**
- typedef struct hess_mc_event_struct **MCEvent**
- typedef struct hess_tel_monitor_struct **TelMoniData**
- typedef struct hess_laser_calib_data_struct **LasCalData**
- typedef struct hess_run_end_statistics_struct **RunStat**
- typedef struct hess_run_end_mc_statistics_struct **MCRunStat**
- typedef struct hess_all_data_struct **AllHessData**

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)

9.33.1 Detailed Description

Definition and structures for H.E.S.S. /CTA data in eventio format.

This file contains definitions and data structures used for writing and reading HESS data (both Monte Carlo and real data) in the eventio format. It was then extended to include potential additional CTA data.

Author

Konrad Bernlöhr

Date

initial version: July 2000

```
CVS $Date: 2014/06/25 12:54:07 $
```

Version

```
CVS $Revision: 1.82 $
```

9.33.2 Macro Definition Documentation

9.33.2.1 #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);
```

Macro expanding into a function call checking if user function.

is taking the same maximum array sizes as the library.

Referenced by main().

9.33.2.2 #define H_MAX_FSHAPE 1000

Max.

number of (sub-) samples of reference pulse shapes.

Referenced by read_hess_pixelset().

9.33.2.3 #define H_MAX_HOTPIX 5

The max.

size of the list of hottest pix.

Referenced by check_hessio_max().

9.33.2.4 #define H_MAX_PIX_TIMES 7

In addition to ADC we may (optionally) also have timing data.

Referenced by pixel_timing_analysis(), and read_hess_pixtime().

9.33.2.5 #define H_MAX_PROFILE 10

The max.

number of MC shower profiles.

Referenced by check_hessio_max(), and read_hess_mc_shower().

9.33.2.6 #define H_MAX_SLICES 128

Maximum number of time slices handled.

Referenced by check_hessio_max(), nb_peak_integration(), print_hess_teladc_samples(), read_hess_teladc_samples(), and write_hess_televent().

9.33.2.7 #define HI_GAIN 0

Which index refers to which type of channel:

Index to high-gain channels in adc_sum, adc_sample, pedestal, ...

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), hesscam_ps_plot(), local_peak_integration(), nb_peak_integration(), read_hess_teladc_sums(), and write_hess_teladc_sums().

9.33.2.8 #define LO_GAIN 1

Index to low-gain channels in adc_sum, adc_sample, pedestal, ...

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), hesscam_ps_plot(), local_peak_integration(), nb-_peak_integration(), read_hess_teladc_sums(), and write_hess_teladc_sums().

9.33.2.9 #define PIX_TIME_PEAKPOS_TYPE 1

Position of peak in time (slices since readout).

Referenced by pixel_timing_analysis().

9.33.2.10 #define PIX_TIME_STARTPOS_ABS_TYPE 3

Position of first rise above absolute threshold.

9.33.2.11 #define PIX_TIME_STARTPOS_REL_TYPE 2

Position of first rise above fraction of peak ampl.

Referenced by pixel_timing_analysis().

9.33.2.12 #define PIX_TIME_WIDTH_ABS_TYPE 5

Width of pulse over absolute threshold (time over threshold).

Referenced by pixel_timing_analysis().

9.33.2.13 #define PIX_TIME_WIDTH_REL_TYPE 4

Width of pulse over fraction of peak ampl.

Referenced by pixel_timing_analysis().

9.33.3 Function Documentation

9.33.3.1 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.

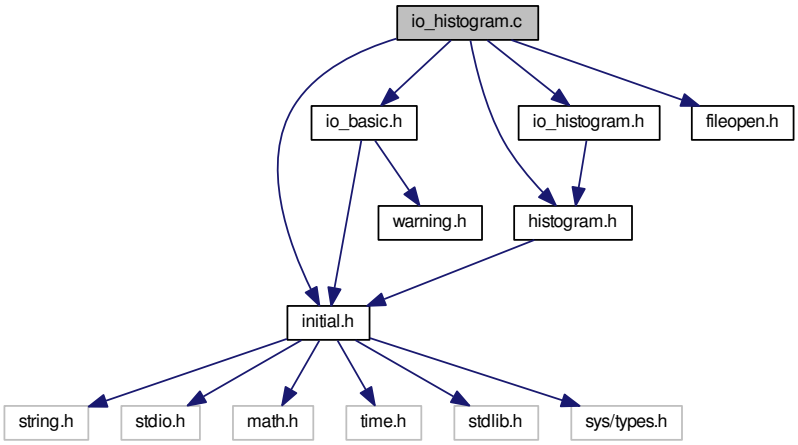
References H_MAX_GAINS, H_MAX_HOTPIX, H_MAX_PROFILE, H_MAX_SLICES, and H_MAX_TEL.

9.34 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:



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.

9.34.1 Detailed Description

This file implements I/O for 1-D and 2-D histograms.

Author

Konrad Bernloehr

Date

1993 to 2010

```
CVS $Date: 2013/10/21 12:53:31 $
```

Version

```
CVS $Revision: 1.20 $
```

9.34.2 Function Documentation

9.34.2.1 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.)

References Histogram_Extension::content_inside, get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_string(), _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.34.2.2 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().

Referenced by main().

9.34.2.3 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.)

References add_histogram(), alloc_2d_int_histogram(), alloc_2d_real_histogram(), alloc_int_histogram(), alloc_real_histogram(), allocate_histogram(), Histogram_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, describe_histogram(), histogram::entries, histogram::extension, Histogram_Extension::fdata, free_histogram(), get_histogram_by_ident(), get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_string(), get_vector_of_long(),

get_vector_of_real(), Histogram_Parameters::integer, histogram::overflow, histogram::overflow_2d, Histogram_-Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, _struct_IO_ITE-M_HEADER::type, histogram::type, histogram::underflow, histogram::underflow_2d, and _struct_IO_ITEM_HEAD-ER::version.

Referenced by read_histograms().

9.34.2.4 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 histogram::counts, histogram::entries, histogram::extension, get_first_histogram(), _struct_IO_ITEM_-HEADER::ident, histogram::ident, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram-::nbins, histogram::nbins_2d, histogram::next, histogram::overflow, histogram::overflow_2d, put_item_begin(), put_-item_end(), put_long(), put_real(), put_short(), put_string(), put_vector_of_long(), put_vector_of_real(), Histogram-_Parameters::real, Histogram_Parameters::sum, histogram::tentries, histogram::title, Histogram_Parameters::tsum, _struct_IO_ITEM_HEADER::type, histogram::type, histogram::underflow, histogram::underflow_2d, Histogram_-Parameters::upper_limit, and _struct_IO_ITEM_HEADER::version.

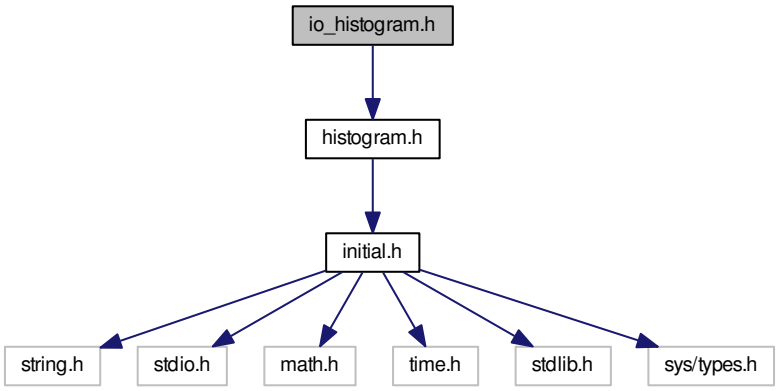
Referenced by main(), write_all_histograms(), and write_dst_histos().

9.35 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)

9.35.1 Detailed Description

Declarations for eventio I/O of histograms.

Author

Konrad Bernloehr

Date

```
CVS $Date: 2013/10/21 12:53:31 $
```

Version

```
CVS $Revision: 1.11 $
```

9.35.2 Function Documentation

9.35.2.1 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.)

References Histogram_Extension::content_inside, get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_string(), _struct_IO_BUFFER::item_length, _struct_IO_BUFFER::item_level, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.35.2.2 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().

Referenced by main().

9.35.2.3 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.)

References add_histogram(), alloc_2d_int_histogram(), alloc_2d_real_histogram(), alloc_int_histogram(), alloc_real_histogram(), allocate_histogram(), Histogram_Extension::content_all, Histogram_Extension::content_inside, Histogram_Extension::content_outside, histogram::counts, Histogram_Extension::ddata, describe_histogram(), histogram::entries, histogram::extension, Histogram_Extension::fdata, free_histogram(), get_histogram_by_ident(), get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_string(), get_vector_of_long(), get_vector_of_real(), Histogram_Parameters::integer, histogram::overflow, histogram::overflow_2d, Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, Histogram_Parameters::tsum, _struct_IO_ITEM_HEADER::type, histogram::type, histogram::underflow, histogram::underflow_2d, and _struct_IO_ITEM_HEADER::version.

Referenced by read_histograms().

9.35.2.4 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 histogram::counts, histogram::entries, histogram::extension, get_first_histogram(), _struct_IO_ITEM_HEADER::ident, histogram::ident, Histogram_Parameters::integer, Histogram_Parameters::lower_limit, histogram::nbins, histogram::nbins_2d, histogram::next, histogram::overflow, histogram::overflow_2d, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), put_string(), put_vector_of_long(), put_vector_of_real(), Histogram_Parameters::real, Histogram_Parameters::sum, histogram::tentries, histogram::title, Histogram_Parameters::tsum, _struct_IO_ITEM_HEADER::type, histogram::type, histogram::underflow, histogram::underflow_2d, Histogram_Parameters::upper_limit, and _struct_IO_ITEM_HEADER::version.

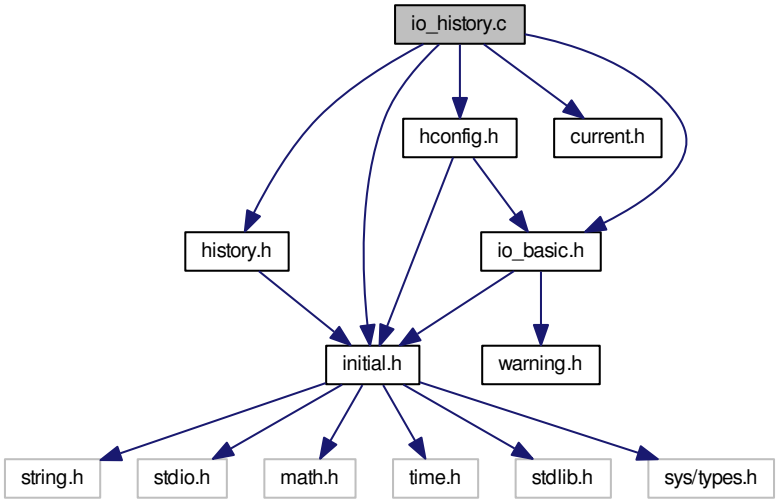
Referenced by main(), write_all_histograms(), and write_dst_histos().

9.36 io_history.c File Reference

Record history of configuration settings/commands.

```
#include "initial.h"
#include "io_basic.h"
#include "history.h"
#include "current.h"
#include "hconfig.h"
```

Include dependency graph for io_history.c:



Data Structures

- struct history_struct

Use to build a linked list of configuration history.

Typedefs

- typedef struct history_struct **HSTRUCT**

Functions

- static void **listtime** (time_t t, FILE *f)
- int **push_command_history** (int argc, char **argv)
- int **push_config_history** (const char *line, int noreplace)
- int **write_history** (long id, IO_BUFFER *iobuf)
- int **write_config_history** (const char *htext, long htime, long id, IO_BUFFER *iobuf)
- int **list_history** (IO_BUFFER *iobuf, FILE *file)

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.

9.36.1 Detailed Description

Record history of configuration settings/commands. This code has not been adapted for multi-threading.

Author

Konrad Bernloehr

Date

1997 to 2010

```
CVS $Date: 2014/02/20 11:40:42 $
```

Version

```
CVS $Revision: 1.8 $
```

9.36.2 Variable Documentation

9.36.2.1 char* cmdline = NULL [static]

A copy of the program's command line.

9.36.2.2 time_t cmdtime [static]

The time when the program was started.

9.36.2.3 HSTRUCT* configs = NULL [static]

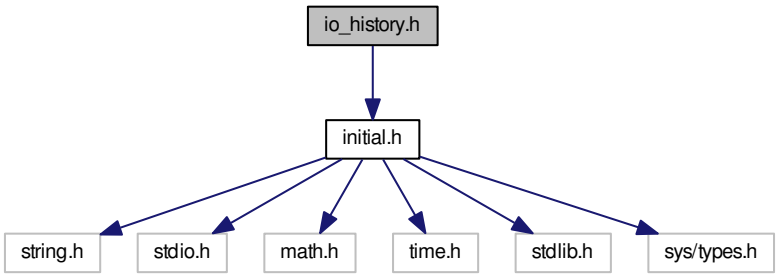
Start of configuration history.

9.37 io_history.h File Reference

Record history of configuration settings/commands.

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

Include dependency graph for io_history.h:



Functions

- int **push_command_history** (int argc, char **argv)
- int **push_config_history** (const char *line, int noreplace)
- int **write_history** (long id, IO_BUFFER *iobuf)
- int **write_config_history** (const char *htext, long htime, long id, IO_BUFFER *iobuf)
- int **list_history** (IO_BUFFER *iobuf, FILE *file)

9.37.1 Detailed Description

Record history of configuration settings/commands.

Author

Konrad Bernloehr

Date

1997 to 2010

```
CVS $Date: 2014/02/20 11:40:42 $
```

Version

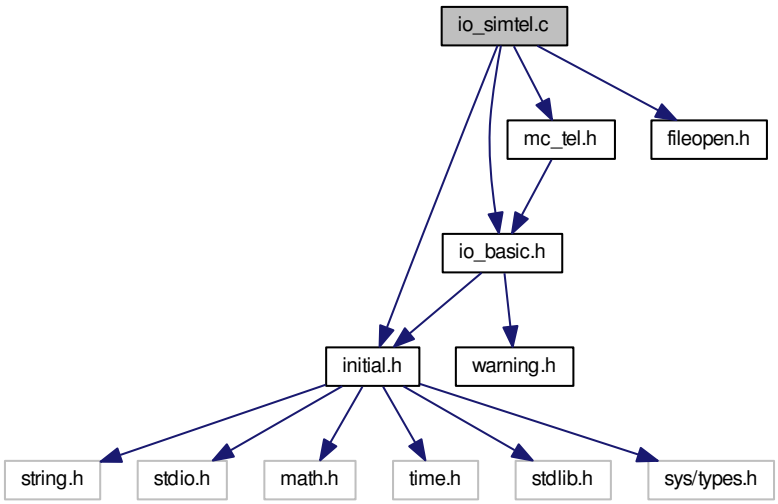
```
CVS $Revision: 1.5 $
```

9.38 io_simtel.c File Reference

Write and read CORSIKA blocks and simulated Cherenkov photon bunches.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "fileopen.h"
```

Include dependency graph for io_simtel.c:



Functions

- int 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_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 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)

  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)

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, struct shower_extra_parameters *ep)
- int **read_shower_extra_parameters** (IO_BUFFER *iobuf, struct shower_extra_parameters *ep)
- int **print_shower_extra_parameters** (IO_BUFFER *iobuf)
- int init_shower_extra_parameters (struct shower_extra_parameters *ep, size_t ni_max, size_t nf_max)

  Initialize, resize, clear shower extra parameters.
- int clear_shower_extra_parameters (struct shower_extra_parameters *ep)

  Similar to init_shower_extra_parameters() but without any attempts to re-allocate or resize buffers.
- struct shower_extra_parameters * **get_shower_extra_parameters** ()

Variables

- static struct shower_extra_parameters 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.

9.38.1 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 2010

```
CVS $Date: 2014/05/07 13:08:25 $
```

Version

```
CVS $Revision: 1.24 $
```

9.38.2 Function Documentation

9.38.2.1 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)

References get_item_begin(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by print_hess_mc_phot(), and read_hess_mc_phot().

9.38.2.2 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.3 int clear_shower_extra_parameters (struct shower_extra_parameters * *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.

References shower_extra_parameters::fparam, shower_extra_parameters::id, shower_extra_parameters::iparam, shower_extra_parameters::is_set, shower_extra_parameters::nfparam, shower_extra_parameters::niparam, and shower_extra_parameters::weight.

Referenced by read_hess_mc_shower().

9.38.2.4 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)

References get_item_end().

Referenced by print_hess_mc_phot(), and read_hess_mc_phot().

9.38.2.5 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)

References put_item_end().

9.38.2.6 int init_shower_extra_parameters (struct shower_extra_parameters * *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.

References shower_extra_parameters::fparam, shower_extra_parameters::id, shower_extra_parameters::iparam, shower_extra_parameters::is_set, shower_extra_parameters::nfparam, shower_extra_parameters::niparam, and shower_extra_parameters::weight.

9.38.2.7 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)

References get_item_begin(), get_item_end(), get_real(), get_short(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.8 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by print_hess_mc_phot().

9.38.2.9 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.38.2.10 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.38.2.11 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 bunch::ctime, bunch::cy, get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), bunch::lambda, bunch::photons, compact_bunch::photons, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, bunch::y, and bunch::zem.

Referenced by main(), and print_hess_mc_phot().

9.38.2.12 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.38.2.13 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)

References get_item_begin(), get_item_end(), get_short(), get_vector_of_real(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.14 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)

References get_item_begin(), get_item_end(), get_long(), get_string(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.38.2.15 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*)

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).

Returns

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

References get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_vector_of_real(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, unget_item(), and _struct_IO_ITEM_HEADER::version.

Referenced by read_hess_mc_phot().

9.38.2.16 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_vector_of_real(), _struct_IO-_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.17 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)

References get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER-::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.18 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)

References get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER-::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.19 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.20 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().

9.38.2.21 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by read_tel_offset().

9.38.2.22 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)

References bunch::ctime, bunch::cy, compact_bunch::cy, get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), bunch::lambda, bunch::photons, _struct_IO_ITEM_HEADER::type, unget_item(), _struct_IO_ITEM_HEADER::version, bunch::y, and bunch::zem.

Referenced by read_hess_mc_phot().

9.38.2.23 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.24 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_short(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.25 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_string(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.26 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*)

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: 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).

Returns

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

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_short(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.27 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.28 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.29 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.30 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.2.31 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)

References compact_bunch::ctime, compact_bunch::cy, fileclose(), fileopen(), _struct_IO_ITEM_HEADER::ident, compact_bunch::lambda, compact_bunch::log_zem, compact_bunch::photons, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, and compact_bunch::y.

9.38.2.32 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().

9.38.2.33 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_real(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by write_tel_offset().

9.38.2.34 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)

References bunch::ctime, bunch::cy, fileclose(), fileopen(), _struct_IO_ITEM_HEADER::ident, bunch::lambda, bunch::photons, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, bunch::y, and bunch::zem.

9.38.2.35 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.38.3 Variable Documentation

9.38.3.1 struct shower_extra_parameters 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.

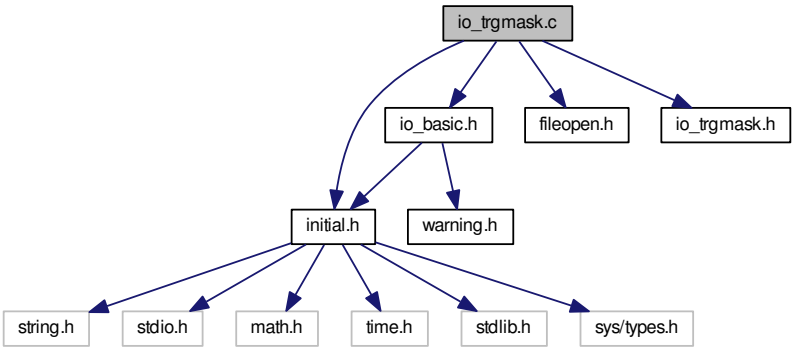
Get a pointer to this block.

9.39 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.

9.39.1 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).

9.39.2 Function Documentation

9.39.2.1 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.

References trgmask_entry::event, trgmask_hash_set::h_e, trgmask_entry::next, and trgmask_entry::tel_id.

Referenced by main(), and merge_data_from_io_block().

9.39.2.2 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.

References trgmask_entry::event, trgmask_hash_set::h_e, trgmask_entry::next, trgmask_entry::tel_id, and trgmask_entry::trg_mask.

9.39.2.3 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.

References trgmask_entry::event, trgmask_hash_set::h_e, trgmask_entry::next, and trgmask_entry::tel_id.

Referenced by check_autoload_trgmask(), main(), and merge_data_from_io_block().

9.39.2.4 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

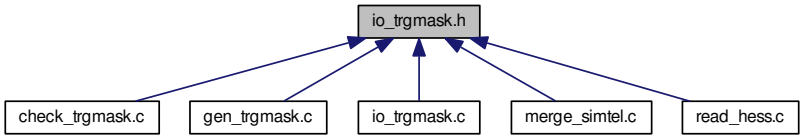
0 (OK), -1 (invalid parameters or file not found), -2 (allocation error, partially filled)

References trgmask_entry::event, fileclose(), fileopen(), trgmask_entry::next, trgmask_entry::tel_id, and trgmask_entry::trg_mask.

9.40 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 **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.

9.40.1 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).

9.40.2 Macro Definition Documentation

9.40.2.1 #define IO_TYPE_HESS_XTRGMASK 2090

Extra (or external - not in normal data file) trigger mask data block type.

Referenced by main(), merge_data_from_io_block(), print_trgmask(), read_trgmask(), and write_trgmask().

9.40.3 Function Documentation

9.40.3.1 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.

References trgmask_entry::event, trgmask_hash_set::h_e, trgmask_entry::next, and trgmask_entry::tel_id.

Referenced by main(), and merge_data_from_io_block().

9.40.3.2 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.

References trgmask_entry::event, trgmask_hash_set::h_e, trgmask_entry::next, trgmask_entry::tel_id, and trgmask_entry::trg_mask.

9.40.3.3 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.

References trgmask_entry::event, trgmask_hash_set::h_e, trgmask_entry::next, and trgmask_entry::tel_id.

Referenced by check_autoload_trgmask(), main(), and merge_data_from_io_block().

9.40.3.4 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)

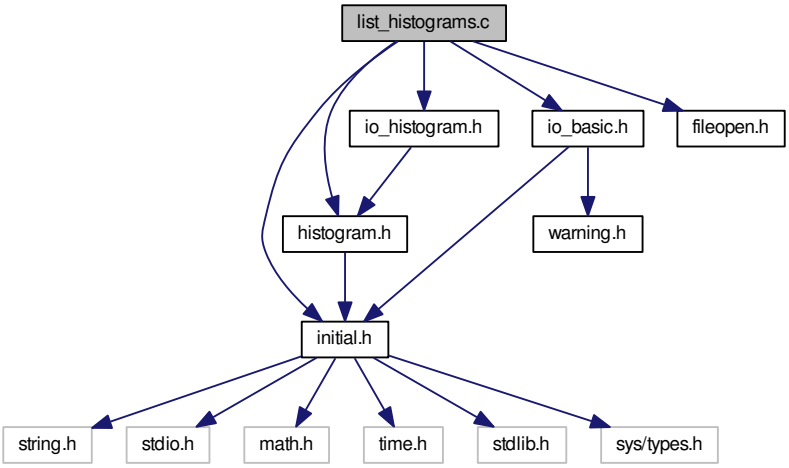
References trgmask_entry::event, fileclose(), fileopen(), trgmask_entry::next, trgmask_entry::tel_id, and trgmask_-entry::trg_mask.

9.41 list_histograms.c File Reference

Utility program for listing histograms.

```
#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

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

  Main program.

9.41.1 Detailed Description

Utility program for listing histograms.

```
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

```
CVS $Date: 2013/10/21 12:53:31 $
```

Version

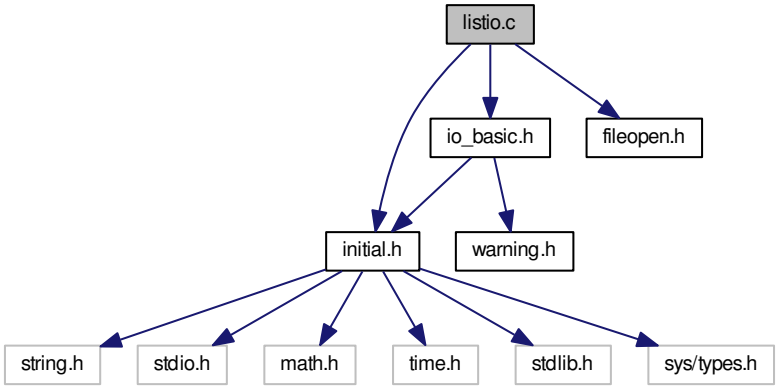
```
CVS $Revision: 1.2 $
```

9.42 listio.c File Reference

Main function for listing data consisting of eventio blocks.

```
#include "initial.h"
#include "io_basic.h"
#include "fileopen.h"
```

Include dependency graph for listio.c:



Functions

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

  Main function.

9.42.1 Detailed Description

Main function for listing data consisting of eventio blocks.

```
@author  Konrad Bernloehr
@date    @verbatim CVS $Date: 2014/06/01 11:33:05 $
```

Version

```
CVS $Revision: 1.14 $
```

The item type, version, length and ident are displayed. With command line option '-s' all sub-items are shown as well. Input is from standard input by default, output to standard output.

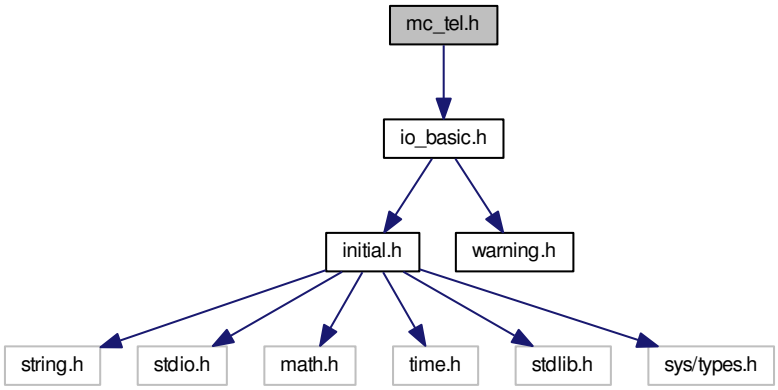
```
Syntax: listio [-s[n]] [-p] [filename]
List structure of eventio data files.
   -s : also list contained (sub-) items
   -sn: list sub-items up to depth n (n=0,1,...)
   -p : show positions of items in the file
If no file name given, standard input is used.
```

9.43 mc_tel.h File Reference

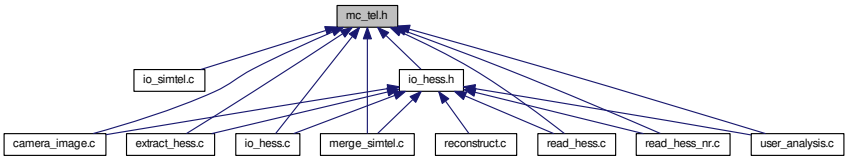
Definitions and structures for CORSIKA Cherenkov light interface.

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

Include dependency graph for mc_tel.h:



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



Data Structures

- struct bunch

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

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

Typedefs

- typedef float **real**
- typedef short **INT16**
- typedef unsigned short **UINT16**
- typedef int **INT32**
- typedef unsigned int **UINT32**

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_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 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)

  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)

  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, struct shower_extra_parameters *ep)
- int **read_shower_extra_parameters** (IO_BUFFER *iobuf, struct shower_extra_parameters *ep)
- int **print_shower_extra_parameters** (IO_BUFFER *iobuf)
- int init_shower_extra_parameters (struct shower_extra_parameters *ep, size_t ni_max, size_t nf_max)

  Initialize, resize, clear shower extra parameters.
- int clear_shower_extra_parameters (struct shower_extra_parameters *ep)

  Similar to init_shower_extra_parameters() but without any attempts to re-allocate or resize buffers.
- struct shower_extra_parameters * **get_shower_extra_parameters** (void)

9.43.1 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 2010

`CVS $Date: 2014/02/20 10:53:06 $`

Version

`CVS $Revision: 1.15 $`

9.43.2 Function Documentation

9.43.2.1 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)

References get_item_begin(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by print_hess_mc_phot(), and read_hess_mc_phot().

9.43.2.2 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.3 int clear_shower_extra_parameters (struct shower_extra_parameters * *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.

References shower_extra_parameters::fparam, shower_extra_parameters::id, shower_extra_parameters::iparam, shower_extra_parameters::is_set, shower_extra_parameters::nfparam, shower_extra_parameters::niparam, and shower_extra_parameters::weight.

Referenced by read_hess_mc_shower().

9.43.2.4 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)

References get_item_end().

Referenced by print_hess_mc_phot(), and read_hess_mc_phot().

9.43.2.5 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)

References put_item_end().

9.43.2.6 int init_shower_extra_parameters (struct shower_extra_parameters * *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.

References shower_extra_parameters::fparam, shower_extra_parameters::id, shower_extra_parameters::iparam, shower_extra_parameters::is_set, shower_extra_parameters::nfparam, shower_extra_parameters::niparam, and shower_extra_parameters::weight.

9.43.2.7 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)

References get_item_begin(), get_item_end(), get_real(), get_short(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.8 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by print_hess_mc_phot().

9.43.2.9 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.43.2.10 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.43.2.11 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 bunch::ctime, bunch::cy, get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), bunch::lambda, bunch::photons, compact_bunch::photons, _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, bunch::y, and bunch::zem.

Referenced by main(), and print_hess_mc_phot().

9.43.2.12 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.43.2.13 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)

References get_item_begin(), get_item_end(), get_short(), get_vector_of_real(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.14 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)

References get_item_begin(), get_item_end(), get_long(), get_string(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by main().

9.43.2.15 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*)

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).

Returns

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

References get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_vector_of_real(), _struct_IO-_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER::type, unget_item(), and _struct_IO_ITEM_HEADER::version.

Referenced by read_hess_mc_phot().

9.43.2.16 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), get_vector_of_real(), _struct_IO-_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.17 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)

References get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER-::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.18 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)

References get_item_begin(), get_item_end(), _struct_IO_ITEM_HEADER::ident, _struct_IO_ITEM_HEADER-::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.19 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), _struct_IO_ITEM_HEADER::type, and _-struct_IO_ITEM_HEADER::version.

9.43.2.20 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().

9.43.2.21 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by read_tel_offset().

9.43.2.22 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)

References bunch::ctime, bunch::cy, compact_bunch::cy, get_item_begin(), get_item_end(), get_long(), get_real(), get_short(), bunch::lambda, bunch::photons, _struct_IO_ITEM_HEADER::type, unget_item(), _struct_IO_ITEM_HEADER::version, bunch::y, and bunch::zem.

Referenced by read_hess_mc_phot().

9.43.2.23 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)

References get_item_begin(), get_item_end(), get_long(), get_real(), get_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.24 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_short(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.25 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_string(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.26 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*)

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: 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).

Returns

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

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_short(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.27 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.28 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.29 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.30 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.43.2.31 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)

References compact_bunch::ctime, compact_bunch::cy, fileclose(), fileopen(), _struct_IO_ITEM_HEADER::ident, compact_bunch::lambda, compact_bunch::log_zem, compact_bunch::photons, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, and compact_bunch::y.

9.43.2.32 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().

9.43.2.33 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)

References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_real(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

Referenced by write_tel_offset().

9.43.2.34 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)

References bunch::ctime, bunch::cy, fileclose(), fileopen(), _struct_IO_ITEM_HEADER::ident, bunch::lambda, bunch::photons, put_item_begin(), put_item_end(), put_long(), put_real(), put_short(), _struct_IO_ITEM_HEADER::type, _struct_IO_ITEM_HEADER::version, bunch::y, and bunch::zem.

9.43.2.35 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)

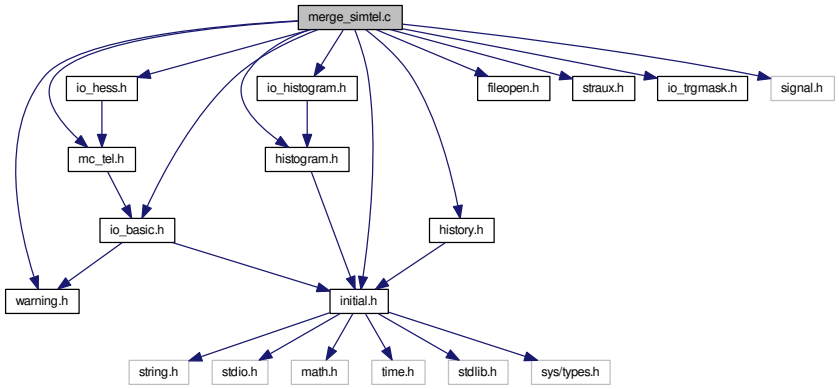
References _struct_IO_ITEM_HEADER::ident, put_item_begin(), put_item_end(), put_long(), put_vector_of_real(), _struct_IO_ITEM_HEADER::type, and _struct_IO_ITEM_HEADER::version.

9.44 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 "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 <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.
- 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, 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 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)
- 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 }

9.44.1 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)
 Once per telescope (and per run for raw & DST levels 0-2; just once for DST level 3):
   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)
 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    @verbatim CVS $Date: 2014/06/25 15:34:16 $
```

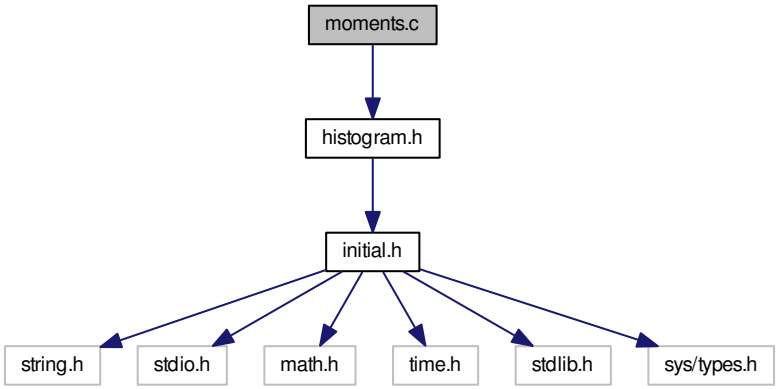
Version

```
     CVS $Revision: 1.4 $
```

9.45 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.

9.45.1 Detailed Description

Calculate mean, rms, skewness, and kurtosis of data.

Author

Konrad Bernloehr

Date

1995 to 2010

Date:

2011/02/28 09:56:42

Revision:

1.3

9.45.2 Function Documentation

9.45.2.1 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().

Referenced by user_init().

9.45.2.2 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(), and user_event_fill().

9.45.2.3 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

9.45.2.4 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

9.45.2.5 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

Referenced by user_event_fill().

9.45.2.6 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

9.45.2.7 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

9.45.2.8 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

9.45.2.9 void free_moments (MOMENTS * *mom*)

Deallocates memory previously allocated to a moments structure.

Parameters

mom	Pointer to previously allocated structure

9.45.2.10 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)

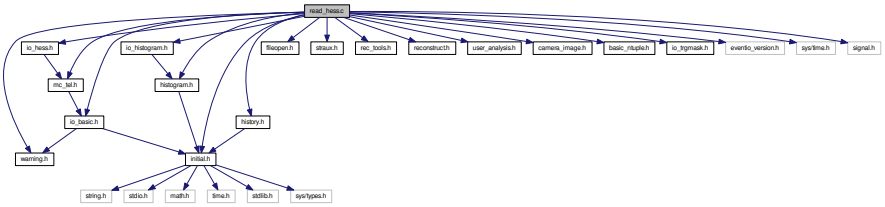
Referenced by user_event_fill().

9.46 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 "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 <sys/time.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.

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

  Main program.

Variables

- struct basic_ntuple **bnt**
- static int **interrupted**
- static int **dst_processing**

9.46.1 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_hess_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).


Syntax: read_hess [ options ] [ - | input_fname ... ]
Options:
   -p ps_filename  (Write a PostScript file with camera images.)
   -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)
   -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-run run1[,run2-run3[,...]] (Select runs being processed.)
   --not-run run1[,run2-run3[,...]]
   --only-telescope id1[,id2-i3[,...]] (Select telescopes being used.)
   --not-telescope id1[,id2-id3[,...]]
   --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.)
   --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.)
   --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.)
   --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).
   --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.)
   --broken-pixels-fraction (Add random broken/dead pixels on run-by-run basis.)
   --dead-time-fraction (Set telescopes randomly as dead from prior triggers.)
   --integration-scheme n *(Set the integration scheme for sample-mode data.)
   --integration-window w,o *(Set integration window width and offset.)
   --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)
   --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.)
   --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.
   --dst-file name (Name of output file for DST-type output.)
   --histogram-file name (Name of histogram file.)
   -f fname        (Get list of input file names from fname.)

Parameters followed by a '*' can be telescope-type-specific if preceded by a
'--type' option. Their interpretation is thus position-dependent.


@author Konrad Bernloehr
```

```
@date    @verbatim CVS $Date: 2014/04/22 15:55:15 $
```

Version

```
     CVS $Revision: 1.108 $


This program started as a skeleton for reading H.E.S.S. data in eventio
format (which is what the read_hess_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).


Syntax: read_hess [ options ] [ - | input_fname ... ]
Options:
   -p ps_filename  (Write a PostScript file with camera images.)
   -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)
   -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-telescope id1[,id2[,...]]
   --not-telescope id1[,id2[,...]]
   --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.)
   --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].)
   --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.)
   --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).
   --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.)
   --auto-lookup   (Automatically generate lookup table (gammas only).)
   --lookup-file name (Override automatic naming of lookup files.)
   --dst-level n   (Level of data reduction when writing DST-type output.)
                   Valid levels: 1, 2, 3, 10, 11, 12, 13.
                   Raw data is stripped off at all levels except 10.
                   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.
   --dst-file name (Name of output file for DST-type output.)
   --dst-process   (Telescope configuration etc. may appear only once.)
   -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    @verbatim CVS $Date: 2010/03/19 18:09:32 $
```

Version

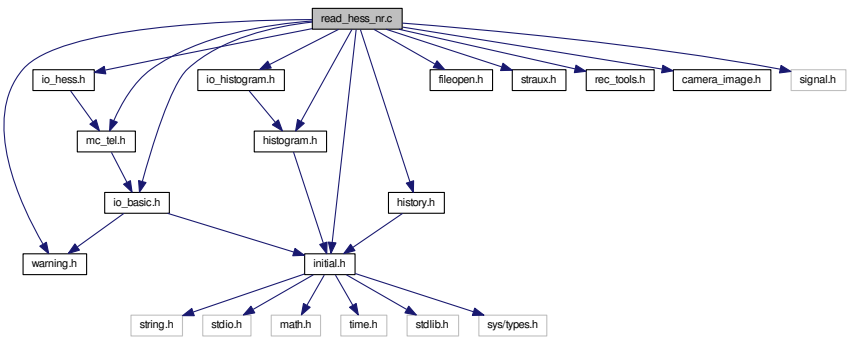
```
    CVS $Revision: 1.76 $
```

9.47 read_hess_nr.c File Reference

A skeleton program reading H.E.S.S.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "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 read_hess_nr.c:



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)
- 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**

9.47.1 Detailed Description

A skeleton program reading H.E.S.S. data.

As a skeleton for programs reading H.E.S.S. data in eventio format, this 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, though, to create nice camera images similar to those generated in sim_hessarray.

```
Syntax: read_hess_nr [ options ] [ - | input_fname ... ]
Options:
   -p ps_filename  (Write a PostScript file with camera images.)
   -r level        (Reconstruction level not fully used in this program version.)
                   level >= 1: show parameters from sim_hessarray.
   -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)
   -i              (Ignore unknown data block types)
   -u              (Call user-defined analysis function)
   --powerlaw x    (Use this spectral index for events weights in output.)
                   (Default spectral index is -2.7)
   --max-events n  (Skip remaining data after so many triggered events.)


@author  Konrad Bernloehr
@date    @verbatim CVS $Date: 2011/07/21 16:07:26 $
```

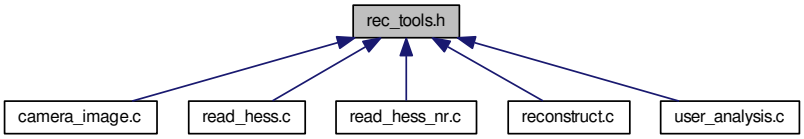
Version

```
     CVS $Revision: 1.16 $
```

9.48 rec_tools.h File Reference

Tools for shower geometric reconstruction.

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.

9.48.1 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, 2009

CVS $Date: 2014/05/07 13:08:25 $

Version

CVS $Revision: 1.17 $

9.48.2 Function Documentation

9.48.2.1 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.

Referenced by main(), shower_reconstruct(), user_event_fill(), and user_init().

9.48.2.2 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(), hesscam_ps_plot(), and user_event_fill().

9.48.2.3 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().

Referenced by shower_geometric_reconstruction().

9.48.2.4 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.

Referenced by shower_geometric_reconstruction().

9.48.2.5 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.

Referenced by shower_geometric_reconstruction().

9.48.2.6 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 main(), mc_event_fill(), second_moments(), user_event_fill(), and user_mc_event_fill().

9.48.2.7 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).

Referenced by cam_to_ref(), and shower_geometric_reconstruction().

9.48.2.8 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.
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.

References cam_to_ref(), get_shower_trans_matrix(), intersect_lines(), and offset_to_angles().

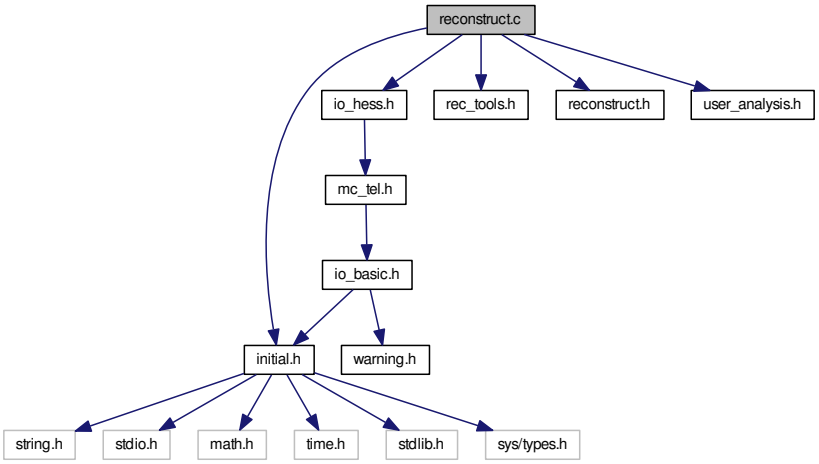
Referenced by shower_reconstruct().

9.49 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 dependency graph for reconstruct.c:



Macros

- #define CALIB_SCALE 0.92

  The factor needed to transform from mean p.e.
- #define **H_MAX_NB1** 8
- #define **H_MAX_NB2** 24

Functions

- 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 have HV disabled or the camera active radius is clipped.
- 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.
- static 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 double **qpol** (double x, int np, double *yval)
- static int **set_integration_correction** (AllHessData *hsdata, int itel, int nbins, int noff)
- 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.
- 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)

  Image/shower reconstruction function.
- void **set_reco_verbosity** (int v)

Variables

- 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]
- 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

9.49.1 Detailed Description

Second moments type image analysis.

Date

```
CVS $Revision: 1.54 $
```

Version

```
CVS $Date: 2014/05/07 13:08:25 $
```

9.49.2 Macro Definition Documentation

9.49.2.1 #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.

Referenced by calibrate_amplitude(), and calibrate_pixel_amplitude().

9.49.3 Function Documentation

9.49.3.1 static int calibrate_amplitude (AllHessData * *hsdata,* int *itel,* int *flag_amp_tm,* double *clip_amp*) [static]

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 hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sum, hess_pixel_timing_struct::after_peak, hess_pixel_timing_struct::before_peak, hess_laser_calib_data_struct::calib, CALIB_SCALE, user_parameters::calib_scale, hess_event_data_struct::central, user_parameters::clip_amp, H_MAX_GAINS, H_MAX_TEL, HI_GAIN, hess_tel_event_adc_struct::known, hess_pixel_timing_struct::known, LO_GAIN, hess_pixel_setting_struct::min_pixel_mult, hess_tel_event_adc_struct::num_gains, hess_camera_settings_struct::num_pixels, hess_central_event_data_struct::num_teldata, hess_central_event_data_struct::num_teltrg, hess_tel_monitor_struct::pedestal, hess_pixel_list::pixel_list, hess_pixel_list::pixels, hess_tel_event_data_struct::pixtm, hess_pixel_timing_struct::pulse_sum_glob, hess_pixel_timing_struct::pulse_sum_loc, hess_tel_event_data_struct::raw, hess_tel_event_adc_struct::significant, hess_tel_event_data_struct::tel_id, hess_event_data_struct::teldata, hess_central_event_data_struct::teldata_list, hess_central_event_data_struct::teldata_pattern, hess_central_event_data_struct::teltrg_list, hess_central_event_data_struct::teltrg_pattern, hess_pixel_timing_struct::threshold, hess_pixel_timing_struct::timval, hess_tel_event_data_struct::trigger_pixels, and user_get_type().

Referenced by image_reconstruct().

9.49.3.2 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.).

References hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sample, hess_tel_event_adc_struct::adc_sum, hess_pixel_timing_struct::after_peak, hess_pixel_timing_struct::before_peak, hess_laser_calib_data_struct::calib, CALIB_SCALE, user_parameters::calib_scale, user_parameters::clip_amp, H_MAX_GAINS, H_MAX_TEL, HI_GAIN, hess_tel_event_adc_struct::known, hess_pixel_timing_struct::known, LO_GAIN, hess_tel_event_adc_struct::num_gains, hess_camera_settings_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_tel_monitor_struct::pedestal, hess_tel_event_data_struct::pixtm, hess_pixel_timing_struct::pulse_sum_glob, hess_pixel_timing_struct::pulse_sum_loc, hess_tel_event_data_struct::raw, hess_tel_event_adc_struct::significant, hess_event_data_struct::teldata, hess_pixel_timing_struct::threshold, hess_pixel_timing_struct::timval, user_get_type(), and hess_tel_event_adc_struct::zero_sup_mode.

9.49.3.3 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.

References H_MAX_TEL, hess_tel_event_data_struct::image_pixels, hess_tel_event_adc_struct::known, hess_camera_settings_struct::num_pixels, hess_pixel_list::pixel_list, hess_pixel_list::pixels, hess_tel_event_data_struct::raw, and hess_event_data_struct::teldata.

Referenced by image_reconstruct().

9.49.3.4 static int find_neighbours (CameraSettings * *camset,* int *itel*) [static]

Find the list of neighbours for each pixel.

References hess_camera_settings_struct::area, hess_camera_settings_struct::num_pixels, hess_camera_settings_struct::size, hess_camera_settings_struct::tel_id, hess_camera_settings_struct::xpix, and hess_camera_settings_struct::ypix.

Referenced by image_reconstruct(), and nb_peak_integration().

9.49.3.5 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).

References hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sample, hess_tel_event_adc_struct::adc_sum, hess_event_data_struct::central, hess_central_event_data_struct::glob_count, H_MAX_TEL, hess_tel_event_adc_struct::known, hess_tel_event_adc_struct::num_gains, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_tel_monitor_struct::pedestal, hess_tel_event_data_struct::raw, hess_tel_event_adc_struct::significant, hess_tel_event_data_struct::tel_id, hess_event_data_struct::teldata, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by pixel_integration().

9.49.3.6 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*) [static]

Calibrate and clean image pixels and reconstruct second moments parameters from images.

References calibrate_amplitude(), clean_image_tailcut(), hess_tel_image_struct::cut_id, find_neighbours(), H_MAX_TEL, hess_tel_event_data_struct::img, hess_tel_event_adc_struct::known, hess_tel_image_struct::known, hess_tel_event_data_struct::num_image_sets, pixel_timing_analysis(), hess_tel_event_data_struct::raw, second_moments(), and hess_event_data_struct::teldata.

Referenced by reconstruct().

9.49.3.7 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 hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sample, hess_tel_event_adc_struct::adc_sum, H_MAX_TEL, HI_GAIN, hess_tel_event_adc_struct::known, LO_GAIN, hess_tel_event_adc_struct::num_gains, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_tel_monitor_struct::pedestal, hess_tel_event_data_struct::raw, hess_tel_event_adc_struct::significant, hess_event_data_struct::teldata, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by pixel_integration().

9.49.3.8 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).
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).

References hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sample, hess_tel_event_adc_struct::adc_sum, find_neighbours(), H_MAX_SLICES, H_MAX_TEL, HI_GAIN, hess_tel_event_adc_struct::known, LO_GAIN, hess_tel_event_adc_struct::num_gains, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_tel_monitor_struct::pedestal, hess_tel_event_data_struct::raw, hess_tel_event_adc_struct::significant, hess_event_data_struct::teldata, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by pixel_integration().

9.49.3.9 static int pixel_integration (AllHessData * *hsdata,* int *itel,* struct user_parameters * *up*) [static]

Pixel integration steering function.

Work is done in selected integration function.

References global_peak_integration(), user_parameters::integ_param, user_parameters::integ_thresh, user_parameters::integrator, local_peak_integration(), nb_peak_integration(), and simple_integration().

Referenced by reconstruct().

9.49.3.10 static int pixel_timing_analysis (AllHessData * *hsdata,* int *itel,* int *nimg*) [static]

Calculate summary results from pixel timing data.

References hess_camera_settings_struct::flen, H_MAX_PIX_TIMES, H_MAX_TEL, hess_tel_event_data_struct::img, hess_pixel_timing_struct::known, hess_tel_event_data_struct::num_image_sets, hess_pixel_timing_struct::num_pixels, hess_pixel_timing_struct::num_types, hess_tel_image_struct::phi, PIX_TIME_PEAKPOS_TYPE, PIX_TIME_STARTPOS_REL_TYPE, PIX_TIME_WIDTH_ABS_TYPE, PIX_TIME_WIDTH_REL_TYPE, hess_tel_event_data_struct::pixtm, hess_event_data_struct::teldata, hess_pixel_timing_struct::time_level, hess_pixel_setting_struct::time_slice, hess_pixel_timing_struct::time_type, hess_pixel_timing_struct::timval, hess_tel_image_struct::tm_residual, hess_tel_image_struct::tm_rise, hess_tel_image_struct::tm_slope, hess_tel_image_struct::tm_width1, hess_tel_image_struct::tm_width2, hess_tel_image_struct::x, hess_camera_settings_struct::xpix, hess_tel_image_struct::y, and hess_camera_settings_struct::ypix.

Referenced by image_reconstruct().

9.49.3.11 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*)

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.

References user_parameters::clip_amp, image_reconstruct(), hess_tel_event_data_struct::img, user_parameters::integrator, hess_tel_event_adc_struct::known, hess_run_header_struct::ntel, pixel_integration(), hess_tel_event_data_struct::raw, shower_reconstruct(), hess_event_data_struct::teldata, and user_get_type().

Referenced by main().

9.49.3.12 static int second_moments (AllHessData * *hsdata,* int *itel,* int *cut_id,* int *nimg,* double *clip_amp*) [static]

Reconstruction of second moments parameters from cleaned image.

References hess_mc_shower_struct::altitude, hess_tel_image_struct::amplitude, hess_mc_shower_struct::azimuth, hess_camera_settings_struct::cam_rot, user_parameters::clip_amp, hess_tel_image_struct::clip_amp, hess_tel_image_struct::cut_id, hess_mc_shower_struct::energy, hess_mc_event_struct::event, hess_camera_settings_struct::flen, H_MAX_TEL, hess_mc_shower_struct::hmax, hess_tel_event_data_struct::img, hess_tel_event_adc_struct::known, hess_tel_image_struct::known, hess_tel_image_struct::kurtosis, hess_tel_image_struct::l, line_point_distance(), hess_tel_event_data_struct::num_image_sets, hess_camera_settings_struct::num_pixels, hess_tel_image_struct::num_sat, hess_tel_image_struct::phi, hess_tel_image_struct::pixels, hess_tel_event_data_struct::raw, hess_tel_image_struct::skewness, hess_camera_settings_struct::tel_id, hess_run_header_struct::tel_pos, hess_event_data_struct::teldata, hess_tel_image_struct::w, hess_tel_image_struct::x, hess_mc_event_struct::xcore, hess_mc_shower_struct::xmax, hess_camera_settings_struct::xpix, hess_tel_image_struct::y, hess_mc_event_struct::ycore, and hess_camera_settings_struct::ypix.

Referenced by image_reconstruct().

9.49.3.13 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)

Referenced by main().

9.49.3.14 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 have HV disabled or the camera active radius is clipped.

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 user_parameters::camera_clipping_deg, hess_camera_settings_struct::flen, H_MAX_TEL, hess_camera_settings_struct::num_pixels, hess_camera_settings_struct::size, which_telescope_type(), hess_camera_settings_struct::xpix, and hess_camera_settings_struct::ypix.

Referenced by main().

9.49.3.15 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 hess_tel_event_adc_struct::adc_known, hess_tel_event_adc_struct::adc_sample, hess_tel_event_adc_struct::adc_sum, H_MAX_TEL, hess_tel_event_adc_struct::known, hess_tel_event_adc_struct::num_gains, hess_tel_event_adc_struct::num_pixels, hess_tel_event_adc_struct::num_samples, hess_tel_monitor_struct::pedestal, hess_tel_event_data_struct::raw, hess_tel_event_adc_struct::significant, hess_event_data_struct::teldata, and hess_tel_event_adc_struct::zero_sup_mode.

Referenced by pixel_integration().

9.50 rndm2.h File Reference

Prototypes for random number generators adapted from HEP Random C++ code.

Macros

- #define rndm(idummy) RandFlat()

  Backwards compatibility with rndm.c.
- #define **rannor**(mean, sigma) RandGauss(mean,sigma)
- #define **rdmin**(iseed) Ranlux_setSeed(iseed,3);
- #define **rdmout**(piseed) fprintf(stderr,"rdmout() not implemented; use Ranlux_getStatus/Ranlux_setStatus instead\n");
- #define **irndm**(idummy) ((long)(RandFlat()*2147483648.))

Typedefs

- typedef int **HepBoolean**
- typedef double(* **PFVD_t**)(void)

Functions

- void **SetRandomEngine** (PFVD_t f)
- void **Ranlux_setSeed** (long seed, int lux)
- void **Ranlux_setSeeds** (long ∗seeds, int lux)
- void **Ranlux_getStatus** (int ∗pseed, int seed_table[24], int ∗pi_lag, int ∗pj_lag, int ∗pcount24, double ∗pcarry)
- void **Ranlux_setStatus** (int ∗pseed, int seed_table[24], int ∗pi_lag, int ∗pj_lag, int ∗pcount24, double ∗pcarry)
- void **Ranlux_saveStatus** (const char ∗fname)
- void **Ranlux_restoreStatus** (const char ∗fname)
- void **Ranlux_showStatus** (void)
- double **Ranlux_RandFlat** (void)
- void **Ranlux_RandFlatArray** (int size, double ∗vect)
- double **RandFlat** (void)
- void **RandFlatArray** (int size, double ∗vect)
- void **RandGauss_setFlag** (HepBoolean val)
- HepBoolean **RandGauss_getFlag** (void)
- void **RandGauss_setVal** (double nextVal)
- double **RandGauss_getVal** (void)
- double **RandGauss** (double mean, double sigma)
- void **RandPoisson_setOldMean** (double val)
- double **RandPoisson_getOldMean** (void)
- double **RandPoisson_getMaxMean** (void)
- void **RandPoisson_setPStatus** (double sq, double alxm, double g)
- double ∗ **RandPoisson_getPStatus** (void)
- long **RandPoisson** (double xm)
- double **RandExponential** (double mean)

9.50.1 Detailed Description

Prototypes for random number generators adapted from HEP Random C++ code.

Author

Konrad Bernloehr

Date

11 July 1997

CVS $Date: 2009/12/07 18:27:28 $

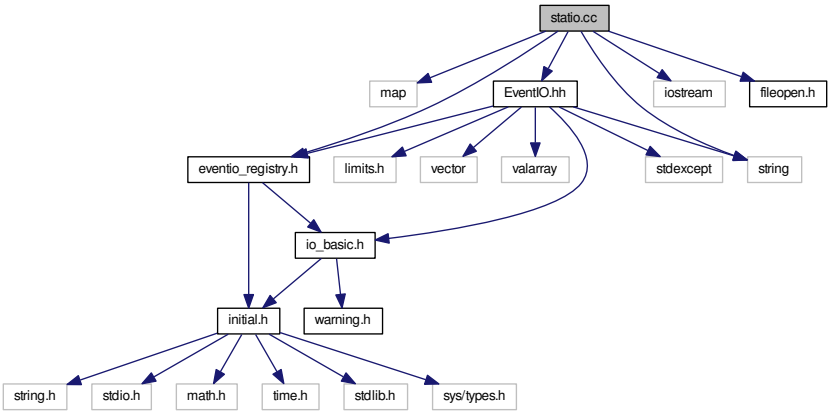
CVS $Revision: 1.5 $

9.51 statio.cc File Reference

A program for statistics of eventio data blocks by block type.

```
#include <map>
#include "EventIO.hh"
#include <iostream>
#include <string>
#include <fileopen.h>
#include <eventio_registry.h>
```

Include dependency graph for statio.cc:



Data Structures

- struct iostats

Macros

- #define **__STDC_LIMIT_MACROS** 1

Functions

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

9.51.1 Detailed Description

A program for statistics of eventio data blocks by block type.

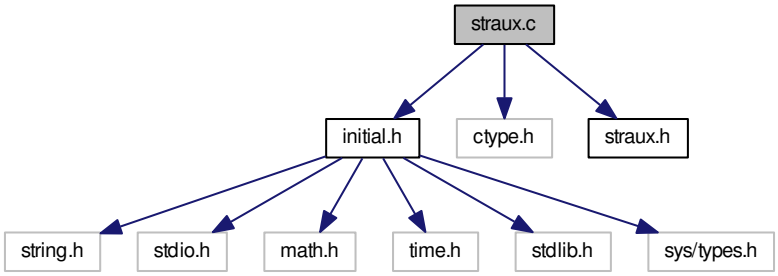
```
Show statistics of EventIO blocks in given files.
Syntax: statio [ -v ] [ -t ] filename [ ... ]
Options:
  -v  Verbose output
  -t  Show total statistics


@author Konrad Bernloehr
```

9.52 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.

9.52.1 Detailed Description

Check for abbreviations of strings and get words from strings.

Author

Konrad Bernloehr

Date

`CVS $Date: 2010/07/20 13:37:45 $`

Version

`CVS $Revision: 1.4 $`

9.52.2 Function Documentation

9.52.2.1 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.

Referenced by do_config(), find_config_item(), and init_config().

9.52.2.2 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 addpath(), do_config(), initpath(), main(), prog_path(), reconfig(), and user_set_tel_type_param_by_str().

9.52.2.3 int stricmp (CONST char * *a*, CONST char * *b*)

Case independent comparison of character strings.

Parameters

a,b	– strings to be compared.

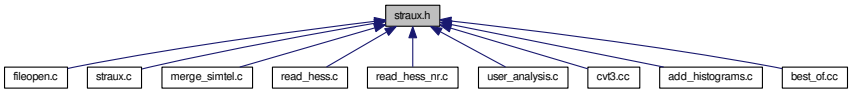
Returns

0 : strings are equal (except perhaps for case) >0 : a is lexically 'greater' than b <0 : a is lexically 'smaller' than b

9.53 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.

9.53.1 Detailed Description

Check for abbreviations of strings and get words from strings.

Author

Konrad Bernloehr

Date

```
CVS $Date: 2010/07/20 13:37:45 $
```

Version

```
CVS $Revision: 1.2 $
```

9.53.2 Function Documentation

9.53.2.1 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.

9.53.2.2 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

9.53.2.3 int stricmp (CONST char * *a*, CONST char * *b*)

Case independent comparison of character strings.

Parameters

a,b	– strings to be compared.

Returns

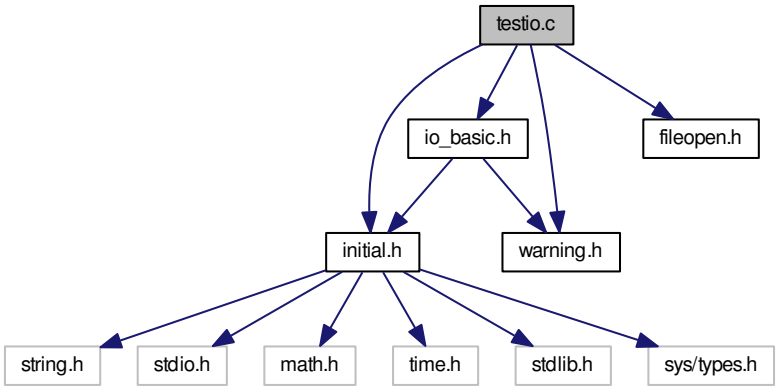
0 : strings are equal (except perhaps for case) >0 : a is lexically 'greater' than b <0 : a is lexically 'smaller' than b

9.54 testio.c File Reference

Test program for eventio data format.

```
#include "initial.h"
#include "warning.h"
#include "io_basic.h"
#include "fileopen.h"
```

Include dependency graph for testio.c:



Data Structures

- struct test_struct

Typedefs

- typedef struct test_struct **TEST_DATA**

Functions

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

  Compare elements of test data structures.
- int write_test1 (TEST_DATA *data, IO_BUFFER *iobuf)

  Write test data with single-element functions.
- int read_test1 (TEST_DATA *data, IO_BUFFER *iobuf)

  Read test data with single-element functions.
- int write_test2 (TEST_DATA *data, IO_BUFFER *iobuf)

  Write test data with vector functions as far as possible.
- int read_test2 (TEST_DATA *data, IO_BUFFER *iobuf)

  Read test data with vector functions as far as possible.
- int write_test3 (TEST_DATA *data, IO_BUFFER *iobuf)

  Write test data in nested items.
- int read_test3 (TEST_DATA *data, IO_BUFFER *iobuf)

  Read test data as a nested tree.
- void syntax (const char *prg)

  Replacement for function missing on OS-9.
- int main (int argc, char **argv)

  Main function for I/O test program.

Variables

- static int **care_long**
- static int **care_int**
- static int **care_short**

9.54.1 Detailed Description

Test program for eventio data format.

Author

Konrad Bernloehr

Date

1994 to 2010

```
CVS $Date: 2014/03/28 15:12:16 $
```

Version

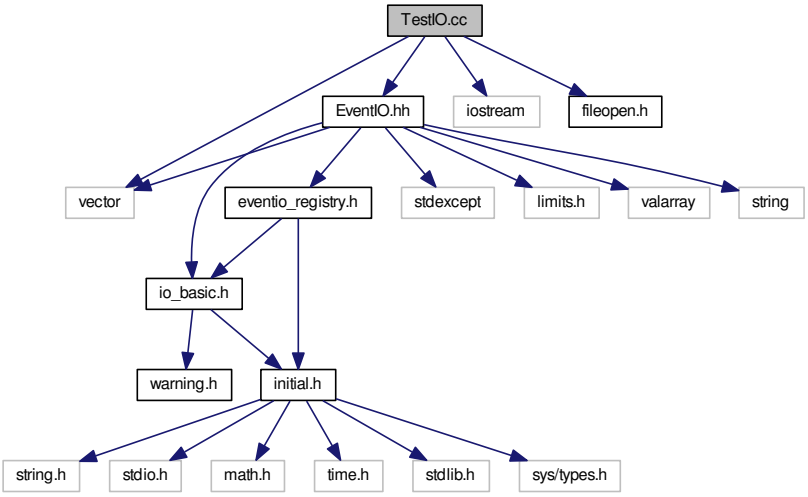
```
CVS $Revision: 1.21 $
```

9.55 TestIO.cc File Reference

Test program for eventio data format (based on testio.c)

```
#include <vector>
#include "EventIO.hh"
#include <iostream>
#include "fileopen.h"
```

Include dependency graph for TestIO.cc:



Data Structures

- struct test_struct

Macros

- #define **__STDC_LIMIT_MACROS** 1

Typedefs

- typedef struct test_struct **TEST_DATA**

Functions

- int datacmp (TEST_DATA &data1, TEST_DATA &data2)

  Compare elements of test data structures.
- int write_test1 (TEST_DATA &data, EventIO &iobuf)

  Write test data with single-element functions.
- int read_test1 (TEST_DATA &data, EventIO &iobuf)

  Read test data with single-element functions.
- int write_test2 (TEST_DATA &data, EventIO &iobuf)

  Write test data with vector functions as far as possible.
- int read_test2 (TEST_DATA &data, EventIO &iobuf)

  Read test data with vector functions as far as possible.
- void **Information** (const char *text)
- void **Warning** (const char *text)
- void **Error** (const char *text)
- int write_test3 (TEST_DATA &data, EventIO &iobuf)

  Write test data in nested items.
- int read_test3 (TEST_DATA &data, EventIO &iobuf)

  Read test data as a nested tree.
- int **write_test_ex** (EventIO &iobuf)
- void **syntax** (const char *prg)
- int main (int argc, char **argv)

  Main function for I/O test program.

Variables

- static int **care_long**
- static int **care_int**
- static int **care_short**

9.55.1 Detailed Description

Test program for eventio data format (based on testio.c) This file is a re-implementation in C++ that should produce the same output (both on screen and in data file) as the original implementation in testio.c. Therefore, this is not intended as a masterpiece in object-oriented programming but a rather straight-forward C-to-C++ translation.

The data files produced both in standard and extended [-e] mode should be identical to the corresponding data files produced with the testio tool. Comparison of the files may serve as an additional test.

Author

Konrad Bernloehr

Date:

2014/03/28 15:12:16

Revision:

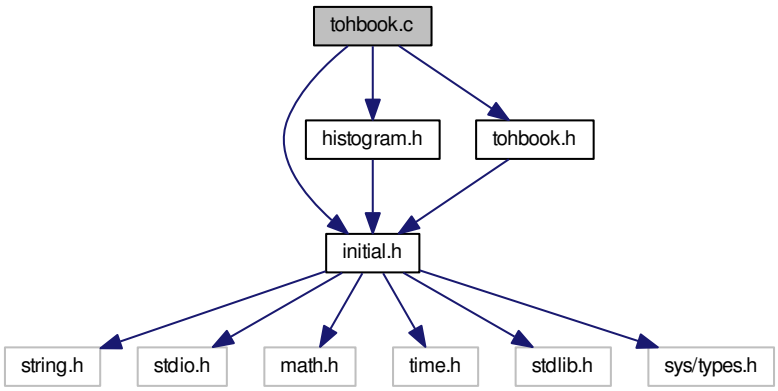
1.23

9.56 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)

9.56.1 Detailed Description

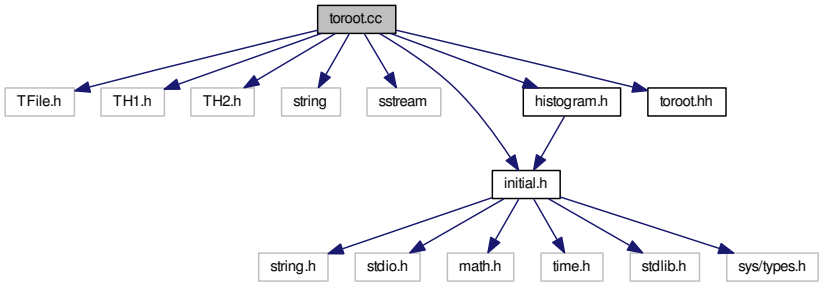
Convert my histograms to HBOOK (PAW) histograms.

9.57 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.

9.57.1 Detailed Description

Functions for conversion of eventio histograms to ROOT format.

Author

Konrad Bernloehr

Date

CVS $Date: 2011/04/15 13:48:04 $

Version

CVS $Revision: 1.12 $

9.57.2 Function Documentation

9.57.2.1 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.

9.57.2.2 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::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.

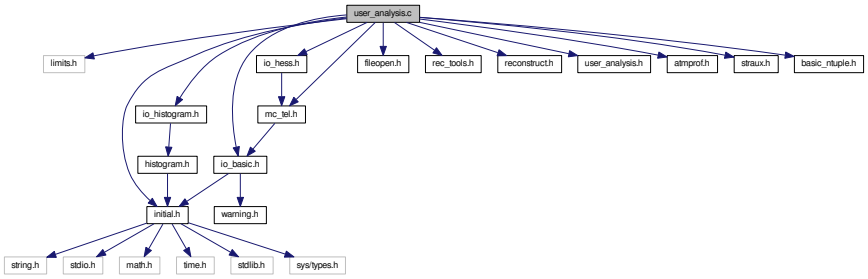
Referenced by convert_histograms_to_root().

9.58 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 "atmprof.h"
#include "straux.h"
#include "basic_ntuple.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

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 struct hess_camera_settings_struct ∗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_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_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)
- void **user_set_integ_threshold** (int ithg, int itlg)
- void **user_set_integ_no_rescale** (int no)
- void **user_set_calib_scale** (double s)
- 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 user_init (AllHessData *hsdata)

  Initialisation of user analysis, booking of histograms etc.
- static void user_mc_shower_fill (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 (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** [1024]
- static char **hist_fname** [1024]
- 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** [1024]

- 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**
- struct basic_ntuple **bnt**

9.58.1 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

`CVS $Date: 2014/05/07 13:08:25 $`

Version

`CVS $Revision: 1.69 $`

9.58.2 Function Documentation

9.58.2.1 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.

References rpol().

Referenced by user_event_fill().

9.58.2.2 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.

Referenced by user_event_fill().

9.58.2.3 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().

Referenced by user_event_fill().

9.58.2.4 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().

9.58.2.5 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.

References Histogram_Extension::ddata, histogram::extension, get_histogram_by_ident(), Histogram_Parameters-::lower_limit, histogram::nbins, histogram::nbins_2d, Histogram_Parameters::real, and Histogram_Parameters-::upper_limit.

Referenced by user_event_fill().

9.58.2.6 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().

9.58.2.7 static char * prog_path (void) [static]

Find the path from which the current program was started.

References getword().

Referenced by user_finish().

9.58.2.8 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().

Referenced by ebias_correction().

9.58.2.9 static void user_done (AllHessData * *hsdata*) [static]

After all data for a file (usually one run) was processed.

9.58.2.10 static void user_event_fill (AllHessData * *hsdata,* int *stage*) [static]

Fill (triggered) event specific histograms etc.

< true energy [TeV]

< 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 basic_ntuple::acceptance, basic_ntuple::alt, hess_shower_parameter::Alt, basic_ntuple::alt_true, hess_mc_shower_struct::altitude, hess_tracking_event_data_struct::altitude_cor, hess_tracking_event_data_struct::altitude_raw, hess_tel_image_struct::amplitude, angle_between(), angles_to_offset(), hess_mc_run_header_struct::atmosphere, basic_ntuple::az, hess_shower_parameter::Az, basic_ntuple::az_true, hess_mc_shower_struct::azimuth, hess_tracking_event_data_struct::azimuth_cor, hess_tracking_event_data_struct::azimuth_raw, calibrate_pixel_amplitude(), hess_event_data_struct::central, basic_ntuple::chi2_e, clear_moments(), user_parameters::clip_amp, hess_tracking_event_data_struct::cor_known, user_parameters::d_sp_idx, ebias_correction(), hess_shower_parameter::energy, hess_mc_shower_struct::energy, hess_shower_parameter::err_dir1, hess_shower_parameter::err_dir2, eval_cut_param(), basic_ntuple::event, hess_mc_event_struct::event, expected_max_distance(), fill_histogram_by_ident(), fill_moments(), hess_mc_shower_struct::h_first_int, H_MAX_TEL, hess_tel_image_struct::hot_amp, hess_tel_event_data_struct::image_pixels, hess_tel_event_data_struct::img, img_norm(), init_atmprof(), hess_tel_event_adc_struct::known, hess_tel_image_struct::known, hess_tel_image_struct::l, basic_ntuple::lg_e, basic_ntuple::lg_e_true, line_point_distance(), basic_ntuple::mdisp, user_parameters::min_amp, user_parameters::min_pix, user_parameters::min_tel_img, hess_shower_parameter::mscl, basic_ntuple::mscrl, basic_ntuple::mscrw, hess_shower_parameter::mscw, basic_ntuple::n_fail, basic_ntuple::n_img, basic_ntuple::n_pix, basic_ntuple::n_trg, basic_ntuple::n_tsl0, hess_run_header_struct::ntel, hess_tel_image_struct::num_hot, hess_tel_event_data_struct::num_image_sets, hess_shower_parameter::num_img, hess_central_event_data_struct::num_teltrg, hess_mc_run_header_struct::obsheight, opt_theta_cut, hess_tel_image_struct::phi, hess_pixel_list::pixel_list, hess_pixel_list::pixels, hess_tel_image_struct::pixels, basic_ntuple::primary, hess_mc_shower_struct::primary_id, hess_tel_event_data_struct::raw, basic_ntuple::rcm, refidx(), hess_shower_parameter::result_bits, basic_ntuple::run, hess_run_header_struct::run, hess_event_data_struct::shower,

basic_ntuple::sig_e, basic_ntuple::sig_mscrl, basic_ntuple::sig_mscrw, basic_ntuple::sig_theta, basic_ntuple::sig_xmax, stat_moments(), hess_tel_event_data_struct::tel_id, hess_run_header_struct::tel_pos, hess_event_data_struct::teldata, basic_ntuple::theta, user_parameters::theta_escale, thickx(), hess_tel_image_struct::tm_residual, hess_tel_image_struct::tm_rise, hess_tel_image_struct::tm_slope, hess_tel_image_struct::tm_width1, hess_tel_image_struct::tm_width2, hess_event_data_struct::trackdata, basic_ntuple::tslope, basic_ntuple::tsphere, user_parameters::user_flags, hess_tel_image_struct::w, basic_ntuple::weight, hess_tel_image_struct::x, basic_ntuple::xc, hess_shower_parameter::xc, basic_ntuple::xc_true, hess_mc_event_struct::xcore, basic_ntuple::xfirst_true, basic_ntuple::xmax, hess_shower_parameter::xmax, hess_mc_shower_struct::xmax, basic_ntuple::xmax_true, hess_tel_image_struct::y, basic_ntuple::yc, hess_shower_parameter::yc, basic_ntuple::yc_true, and hess_mc_event_struct::ycore.

9.58.2.11 static void user_finish (AllHessData * *hsdata*) [static]

Final call before program terminates.

References hess_mc_shower_struct::primary_id, prog_path(), and write_all_histograms().

9.58.2.12 int user_get_type (int *itel*)

Get the best matching telescope type for a given telescope index.

If user analysis is not activated, this will always be type 0.

References H_MAX_TEL.

Referenced by calibrate_amplitude(), calibrate_pixel_amplitude(), main(), and reconstruct().

9.58.2.13 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 hess_mc_shower_struct::altitude, hess_mc_shower_struct::azimuth, user_parameters::d_sp_idx, hess_mc_shower_struct::energy, fill_histogram_by_ident(), line_point_distance(), hess_mc_event_struct::xcore, and hess_mc_event_struct::ycore.

9.58.2.14 static void user_mc_shower_fill (AllHessData * *hsdata*) [static]

Work to be done once per generated shower.

9.58.2.15 void user_set_clipping (double *dc*)

Set the maximum radius to be used of a camera.

References user_parameters::camera_clipping_deg.

Referenced by main().

9.58.2.16 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.

References user_parameters::user_flags.

Referenced by main().

9.58.2.17 void user_set_length_max_cut (double * *lmax*)

Set energy dependent scaled length limit.

Referenced by main().

9.58.2.18 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().

Referenced by main().

9.58.2.19 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.

References user_parameters::theta_escale.

Referenced by main().

9.58.2.20 void user_set_width_max_cut (double * *wmax*)

Set energy dependent scaled width limit.

Referenced by main().

9.58.3 Variable Documentation

9.58.3.1 double opt_theta_cut[7][H_MAX_TEL] [static]

Angular cut limit is multiplicity dependent.

Referenced by user_event_fill(), and user_init().

9.58.3.2 int telescope_type[H_MAX_TEL] [static]

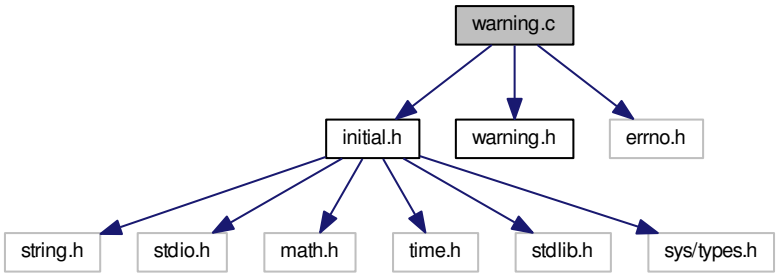
Declare local (static) data here ...

9.59 warning.c File Reference

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

```
#include "initial.h"
#include "warning.h"
#include <errno.h>
```

Include dependency graph for warning.c:



Data Structures

- struct warn_specific_data

  A struct used to store thread-specific data.

Macros

- #define **__WARNING_MODULE** 1
- #define **get_warn_specific**() (&warn_defaults)

Functions

- void 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 auxilliary function for warnings.
- void **set_default_aux_warning_function** (char ∗(∗auxfunc)(void))

Variables

- static struct warn_specific_data **warn_defaults**

9.59.1 Detailed Description

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

```
@author  Konrad Bernloehr
@date    @verbatim CVS $Date: 2014/02/20 10:53:06 $
```

Version

```
CVS $Revision: 1.9 $
```

One of the most import parameter for setting up the bevaviour is the warning level:

```
-------------------------------------------------------------
 Warning level: The lowest level of messages to be displayed
-------------------------------------------------------------
 Warning mode:
 bit 0: display on screen (stderr),
 bit 1: write to file,
 bit 2: write with user-defined logging function.
 bit 3: display origin if supplied.
 bit 4: open log file for appending.
 bit 5: call auxilliary function for time/date etc.
 bit 6: use the auxilliary function output as origin string
        if no explicit origin was supplied.
 bit 7: use syslog().
-------------------------------------------------------------
```

9.59.2 Function Documentation

9.59.2.1 void flush_output (void)

Flush buffered output.

Output is flushed, no matter if it is standard output or a special output function;

Returns

(none)

Referenced by set_output_function().

9.59.2.2 void set_aux_warning_function (char ∗(∗)(void) *auxfunc*)

Set an auxilliary 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)

9.59.2.3 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)

References warn_specific_data::logfname.

9.59.2.4 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)

9.59.2.5 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)

References flush_output().

9.59.2.6 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.

9.59.2.7 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)

9.59.2.8 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)

References warn_specific_data::logfname.

9.59.2.9 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)

9.59.3 Variable Documentation

9.59.3.1 struct warn_specific_data warn_defaults [static]

Initial value:

```
=
{
   0,
   1+8,
   "",
   "warning.log",
   "",
   0,
   NULL,
   NULL,
   NULL,
   NULL,
   0
}
```

9.60 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 auxilliary function for warnings.
- void **set_default_aux_warning_function** (char ∗(∗auxfunc)(void))
- char ∗ **warn_f_get_message_buffer** (void)

9.60.1 Detailed Description

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

Author

Konrad Bernloehr

Date

`CVS $Date: 2010/07/20 13:37:45 $`

Version

`CVS $Revision: 1.5 $`

9.60.2 Function Documentation

9.60.2.1 void flush_output (void)

Flush buffered output.

Output is flushed, no matter if it is standard output or a special output function;

Returns

(none)

Referenced by set_output_function().

9.60.2.2 void set_aux_warning_function (char ∗(∗)(void) *auxfunc*)

Set an auxilliary 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)

9.60.2.3 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)

References warn_specific_data::logfname.

9.60.2.4 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)

9.60.2.5 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)

References flush_output().

9.60.2.6 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.

9.60.2.7 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)

9.60.2.8 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)

References warn_specific_data::logfname.

9.60.2.9 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