CORSIKA add-on package IACT/ATMO: Reference Manual

Version 1.33

Generated by Doxygen 1.4.6

Fri Apr 7 13:38:48 2006

Contents

1 Introduction

Introduction to the IACT/ATMO package

This is the README file for the CORSIKA supplements for Cherenkov light by Konrad Bernlöhr. It was last updated for release 1.35 (April 2006).

The Software and data files provided with this package are intended to enhance the CORSIKA air shower simulation program by

a) tabulated atmospheres for different climate zones, including accurate indices of refraction and

b) a flexible interface for arbitrary configurations of Cherenkov light detectors, which don't need to be in a horizontal plane. Although intended for systems of Imaging Atmospheric Cherenkov Telescopes (IACTs), it can be used for any kind of Cherenkov detectors.

c) A machine-independent output format for the 'photon bunches' collected at each of the assumed telescopes or other detector (format and relevant software termed 'eventio').

The main CORSIKA distribution comes with suitable function interfaces which can be selected in the CMZ extraction step. See the IACT and ATMEXT options in the CORSIKA User's Guide. For successfully compiling and linking CORSIKA when either or both of these options is selected, you need the software provided here.

Note that depending on other CORSIKA options selected, compilation of some of the files provided here might require different preprocessor flags. If your are using some 5.8xx or 5.9xx beta version of CORSIKA you should also carefully check that the interfaces match, since they have changed in the development phase. If linking with earlier versions set the '`-DCORSIKA_VERSION=...`' preprocessor definition as appropriate, for example '`-DCORSIKA_VERSION=5900`'. Interfaces have been stable since version 5.901 (up to and including 6.032, as of now). For the 6.2xx series an internal change to the random number generation has beeen applied. When building the CORSIKA program with GNU make (that means through the supplied `GNUmakefile`), automatic version detection is applied and no manual definitions should be needed. For the 6.5xx series the supplied `GNUmakefile` is not applicable. The build process of CORSIKA 6.5xx also covers the IACT option.

Part a) is implemented in '`atmo.c`' and includes files `atmprof1.dat`, `atmprof2.dat`, `atmprof3.dat`, `atmprof4.dat`, `atmprof5.dat`, and `atmprof6.dat`. Files `atmprof[1-6].dat` contain atmospheric profiles as tabulated in

- F.X.Kneizys et al. (1996), The MODTRAN 2/3 report and LOWTRAN 7 model, Phillips Laboratory, Hanscom AFB, MA 01731, U.S.A.

Indices of refraction were calculated with the effect of water vapour taken into account. An additional file `atmprof9.dat` was constructed for the antarctic winter climate from publicly available radiosonde data for several antarctic stations, in particular the Amundsen-Scott South Pole Station. Beyond 32 km altitude this atmospheric profile is based on extrapolations which might not be very accurate.

Part b) is implemented in its most basic form in '`iact.c`' but for taking full use of it, additional software ('eventio') is required. Instead of a rectangular grid of rectangular detectors, as in previous CORSIKA implementations, individual detectors can be placed individually. Positions don't need to be in a horizontal plane but may stick out of the lowest CORSIKA detection layer. No detector details are needed in the CORSIKA run except for the radius of a sphere containing each detector. Data is recorded in the usual form of CORSIKA 'photon bunches'. With the additional software (see part c) these are recorded in a machine-independent format for later detector and atmospheric transparency simulation. An example skeleton program for such follow-on processing is included in this distribution.

Output in the 'eventio' data structure can be either written into a file or passed directly (piped) to another program. To use such a pipe for immediate processing of the output (from its standard input), use a leading '|' character (perhaps following a '+' if long format is desired) with the TELFIL keyword in the CORSIKA input. Note that no blanks can be part of the file name or pipe supplied due to CORSIKA input processing. The length is also rather limited and any nonprintable or shell metacharacters are forbidden for pipes. If you need argument lists or further redirection, use a shell script for that purpose.

If the output file name ends in `.gz_` or `.bz2_`, then files will be compressed on the fly with the gzip or bzip2 programs, respectively. This typically saves on the order of 30% disk space. These compressed files can also be read on the fly in your analysis program if you replace the fopen() call with fileopen() and the fclose() call with fileclose().

Note also that the TELFIL parameter may contain up to 7 integers appended to the actual name, with colons ':' as separators. Values 1 to 5 determine the amount of information written to standard output for each or every n'th event, while value 6 determines when photon bunches stored in main memory are swapped to temporary files. See the `telfil_` function in `iact.c` for details. Value 7 may be used to override the maximum size for the I/O buffer.

1: number of events for which the photons per telescope are shown
2: number of events for which energy, direction etc. are shown
3: every so often an event is shown (e.g. 10 -> every tenth event).
4: every so often the event number is shown even if no. 1 and no. 2 ran out.
5: offset for no. 4 (no. 4=100, no. 5=1: show events 1, 101, 201, ...)
6: the maximum number of photon bunches before using external storage
7: maximum I/O buffer size in Megabytes.
Example: TELFIL +iact.dat:5:15:10

Depending on whether your CORSIKA writes Cherenkov light vertical distributions as integral distributions or as light emission distributions you might need to set a '`-DINTEGRATED_LONG_DIST`' preprocessor definition when compiling '`iact.c`'. You will need that if '`corsika.car`' does NOT contain the string '`INTCLONG`'. You will also need it if you have a newer CORSIKA and you CMZ extracted it with the backwards- compatibility option INTCLONG. With the '`-DINTEGRATED_LONG_DIST`' flag, the Cherenkov light distribution obtained from CORSIKA will be differentiated before writing it. All other vertical distributions are unaffected.

Part c) This additional software referred to as 'eventio' provides a rather general means for machine-independent I/O although with a restricted set of basic data types (implemented in '`eventio.c`' and '`io_basic.h`', requiring also '`warning.c`' and '`warning.h`') and additional files providing higher-level functions for the IACT interface ('`io_simtel.c`', '`mc_tel.h`', '`initial.h`').

The eventio buffers have a user-defined maximum size to avoid that your system gets into trouble when a shower with an unexpectedly large number of photon bunches is encountered. This maximum has a compile-time default, normally 200 Megabytes, that you can change by 'make MAX_IO_BUFFER=500000000', for example, and which you can also override at run-time with the optional value number 7 of the TELFIL option (see above).

Please note the copyright notices (in file 'Copyright' and in individual source files). If you want to use the software for other purposes than intended (i.e. with CORSIKA), just ask me (Konrad.Bernloehr@mpi-hd.mpg.de).

Installation notes

Recent versions of this software were mainly tested under Linux (on i386 and x86_64 architectures) with GNU g77 and gcc version 3.2 to 3.4.3. On the x86_64 (AMD Opteron) architecture, both 32 and 64 bit modes are possible. The supplied GNUmakefile fill prefer the 64 bit mode.

Note that with GCC 4.0.0 the g77 compiler was replaced by gfortran, a Fortran95 compiler which, at least up to version 4.0.1, does not (yet) compile the CORSIKA and interaction model codes - mainly due to multiple 'SAVE' statements. The incompatibility was classified as a 'regression' by GCC maintainers and will hopefully be fixed in the next GCC version. The C code provided with this package has no problems with gcc 4.0.0/4.0.1. And there are no problems by mixing gcc 4.0.x as the C compiler with g77 from earlier releases. This has been tested with gcc 4.0.1 and g77 3.3.6.

Older CORSIKA versions were also tested under DEC UNIX 4.0 ('OSF/1'), compiled with DEC f77 and cc and also with GNU g77 and gcc. The older versions were also tested under Linux (i386) with GNU g77 and gcc (egcs 1.1.2 and gcc 2.7.2, 2.95.2, 2.96, and most 3.x versions). However, for gcc versions 2.95.2 and earlier we have seen what appear to be code-generation bugs when compiling Fortran files with optimization enabled, resulting in Not-a-Number values leading in turn to endless loops. If that happens to you, try compiling with '-O0' (optimization disabled). With later gcc versions (2.96 and 3.0.x to 3.4.3 at this time), no such bugs were seen and full optimization appears to be fine.

It is expected to run also on other flavours of UNIX, at least with gcc/g77 and perhaps also under some of the other more or less popular 'operating systems' for Intel x86/Pentium PCs, but I never tried. On anything more exotic (from my point of view) I will probably not be able to give any help for porting this software. One area of potential trouble will be interfacing FORTRAN and C functions which is implemented here in the common way on UNIX systems, with an underscore added at C functions called from FORTRAN. All parameters are passed by address, system-specific passing of character string lengths is circumvented by null-terminating the strings before passing them to C functions.

The 'eventio' software has been tested and is in use under a wider variety of operating systems and CPUs. A test program is included with this distribution and you are advised to compile and run the test program first, before trying out the higher-level software. Output of this 'testio' test program might look like

```
Write test data to file 'test.dat'.
Default byte order, using mainly vector functions.
Default byte order, using single-element functions.
Reversed byte order, using single-element functions.
Normal byte order, using single-element functions.
Write tests done.

Read test data from file 'test.dat'.
Default byte order, using single-element functions.
Default byte order, using mainly vector functions.
Reversed byte order, using single-element functions.
Normal byte order, using single-element functions.
Read tests done

Everything is ok. Congratulations!
```

```
Note: on this machine you should care about the sign propagation
of 'LONG' data elements to long integer variables.

Note: on this machine you should care about the sign propagation
of 'SHORT' data elements to integer or long integer variables.
```

For a very brief introduction to eventio see the comments at the beginning of '`eventio.c`'. A detailed description is available, both in English and in German. Most likely, the basics of 'eventio' should be of less interest to you than the interfaces to the higher-level functions described in comments in '`io_simtel.c`' and the skeleton of a telescope simulation program included with this distribution (in `sim_skeleton.c`). Selected source code comments, function prototypes and cross references of function calls as obtained with the `doxygen` tool are contained in '`iact_refman.pdf`'.

Note that structured 'eventio' data blocks can be listed with the 'listio' tool also provided with this distribution.

For your convenience, a file '`GNUmakefile`' is included in the distribution to ease building of the programs. This file requires GNU make to work. Targets in the GNUmakefile include CORSIKA itself as well as the 'testio' and 'listio' programs. With GNU make, there is a very good chance that no modifications to '`GNUmakefile`' are needed.

As a first step, check that 'eventio' works:

```
make test
```

That should compile and link the 'testio' program and run the basic 'eventio' tests. You might also want to

```
make list
```

which creates the 'listio' program and shows the structure of the test data file created.

For creating the CORSIKA executable through GNU make (with the GNUmakefile file), you should have the CMZ source file available as '`corsika.car`' and you should also have source files for the hadronic interaction models available as `gheisha2002.f` and `qgsjet01c.f` (or `venus.f`). The GNUmakefile takes advantage of a number of GNU make features and will test for variants of the interaction model sources as available with earlier versions of CORSIKA. It should therefore also work with older version of CORSIKA. If several CORSIKA versions are available in the same directory (`corsika6xxx.car`), the highest version number will be used by default.

For extracting a requested variant of CORSIKA, a UNIX 'csh' shell script '`cmz_extract`' is included, working with '`GNUmakefile`'. This script can be used instead of the CMZ 'make' macro which can only be used interactively. The default CORSIKA version built is with the CERENKOV, IACT, VIEWCONE, VOLUMEDET and ATMEXT options. Default interaction models are now QGSJET and Gheisha. The GNUmakefile will automatically detect if you have QGSJET01c or QGSJET01 and adjust CMZ options accordingly. Under most UNIX-like systems building CORSIKA should be as simple as

```
make
```

(after compiler-specific flags have been adapted in the Makefile or when using GNU make). Note that building CORSIKA with this Makefile will work only on UNIX-like systems. On other systems you should either extract CORSIKA from CMZ by the interactive method described in the CORSIKA User's Guide or extract it first on a UNIX machine.

If you decide to build different versions of CORSIKA, running

```
make clean
```

inbetween will help to make sure that the parts fit together. But even if you don't clean up inbetween, it should usually be OK (UNIX required). You could, for example, then

```
make INTERACTION=venus OPTIONS="urqmd atmext"
```

to build a version of CORSIKA without Cherenkov light production but with the extension for tabulated atmospheric profiles, and with VENUS high-energy model and the URQMD low-energy model. Note, however, that '`make`' will not catch changes in preprocessor definitions, e.g. as in

```
make INTERACTION=gqsjet DEFINES=-DINTEGRATED_LONG_DIST
```

That is where a '`make clean`' is really appropriate. Changes in CORSIKA's CMZ options will be catched though, that is after a

```
make OPTIONS="viewcone volumedet iact atmext"
```

to build the program `corsika` and a later

```
make EXTRA_OPTIONS=ceffic CORSIKA=corsika_ceffic
```

to build a second program `corsika_ceffic`. Note that the `EXTRA_OPTIONS` here are added to the default options while with `OPTIONS` you would override the default options. The `CORSIKA` variable in the example above is used to build the resulting program under an alternative name. See the CORSIKA User's Guide for CMZ options and required additional files if you want to add built other variants of CORSIKA.

The iact/atmo package may also contain some 'patches' in the unified diff format which have not yet been merged into the mainline CORSIKA distribution. Apart from fixes for known problems, these currently include shorter step-lengths for muons and electrons, as appropriate to Cherenkov telescopes with pixel sizes of less than 0.1 degrees, and extensions to the IACT interface as activated through the CMZ option 'IACTEXT'. These patches can be applied to the original CORSIKA version. For example, if you have version 6.204 of CORSIKA and this package includes a 'corsika6204e.patch file, you can apply this patch through

```
patch -p1 < corsika6204e.patch
```

The extensions are not activated by default but can be activated by the following CMZ flags (i.e. by adding them to the `EXTRA_OPTIONS=...` assignment for make):

CERWLEN The index of refraction is made wavelength dependent. As a consequence, photon bunches will carry a specific wavelength. Photons of shorter wavelengths (with larger index of refraction) will result in larger Cherenkov cone opening angles and larger bunch sizes. For very fast particles this will generally have a small effect (less than 0.03 deg in the opening angle, for example) but near the Cherenkov threshold the effect can be larger.
This option may also require to use a smaller maximum bunch size (see CERSIZ keyword) since all photons in a bunch are of the same wavelength and, therefore, the peak quantum efficiency rather than the average quantum efficiency determines the maximum acceptable bunch size. (In combination with the CEFFIC option, you should use a maximum bunch size of 1, as usual.)

IACTEXT The interface to the `TELOUT` function is extended by parameters describing the emitting particle. This extended information is stored as an additional photon bunch (after the normal one) with mass, charge, energy, and emission time replacing the `cx`, `cy`, `photons`, and `zem` fields, respectively, and are identified by a wavelength of 9999. The compact output format is disabled for making that possible. In addition, all particles arriving at the CORSIKA observation level are included in

the eventio format output file, in a photon-bunch like block identified by array and detector numbers 999.
The `x`, `y`, `cx`, `cy`, and `ctime` fields keep the normal sense, with coordinates, directions and time counted in the CORSIKA detection level reference frame. The particle momentum is filled into the `zem` field (negative for upward-moving particles) and the particle ID is filled into the `lambda` field. If thinning is used, the particle weight is in the `photons` field.
When compiling iact.c manually (instead of taking advantage of the GNUmakefile), an additional option `-DIACTEXT` is required to have a consistent interface on both sides.

You probably obtained this file through the CORSIKA download area (see `http://www-ik.fzk.de/corsika/` for instructions). The version there is not updated very often. For more frequent updates to the iact/atmo package see also `http://www.mpi-hd.mpg.de/hfm/~bernlohr/iact-atmo/`, where you will need to identify as user '`iact`', password '`corsika`'.

This ReadMe file, the documentation in the other files of this distribution, and in the CORSIKA User's Guide may answer most of your questions but perhaps not all. In case of further questions or installation problems you are welcome to send e-mail to me, i.e. `Konrad.Bernloehr@mpi-hd.mpg.de` Questions entirely related to CORSIKA itself and neither related to the Cherenkov light emission in CORSIKA nor to the extensions provided with this distribution should be better directed to Dieter Heck, i.e. `heck@ik.fzk.de`

If you manage to get the enhancements running on other machines than used by me, I would be glad to hear from you. Other users might gain from your experience.

Usage notes

Finally, I would like to add a few notes on the usage of CORSIKA for Cherenkov light simulations:

a) If using the LONGI keyword for sampling the 'longitudinal' (vertical) shower development and either taking a CORSIKA version before June 2000 or a later one extracted with INTCLONG option, then of the order of 40% of the CPU time is actually spent in functions CERLDE and CERLDH which are extremely inefficient in their backwards compatible form. If using them that way, the file '`iact.c`' should be compiled with the preprocessor definition '`-DINTEGRATED_LONG_DIST`' to differentiate the Cherenkov light vertical distribution. If you think that you need the longitudinal development for shower particles but not for Cherenkov light and have a pre- June 2000 CORSIKA version, you are advised to remove the calls to CERLDE and CERLDH in subroutine CERENK. With newer versions, this can be accomplished by the NOCLONG option when extracting from CMZ. The default behaviour of new CORSIKA versions should be to sum up photon information only at the level of emission.

b) The Cherenkov photons are stored in 'bunches'. CORSIKA includes an automatic calculation of the bunch size, depending on primary energy. However, this calculation was only optimised for the non-imaging Cherenkov counters (AIROBICC) in the HEGRA array. For Cherenkov telescopes this value will usually be too high. Therefore, you are advised to set an appropriate bunch size for your purposes with the CERSIZ keyword. For imaging telescopes with pixels sensitive at the one photo-electron level, using conventional photomultiplier tubes, a bunch size of 5 to 10 photons seems appropriate. Larger bunch sizes would add artificial 'noise' to your images because, in simple words, you either measure no photo-electron in a pixel or you measure several photo-electrons when the pixel is hit by just one bunch.

   When using the CEFFIC option, where atmospheric absorption, mirror reflectivity and quantum efficiency are already applied in CORSIKA, the bunch size has to be reduced even further. In that case, a bunch size of 5 would mean that, typically, you either measure no or five (or a multiple of about five) photo-electrons. A bunch size of 1 (photo-electron) appears more appropriate then.

Note: *The CORSIKA data files provided for use with the CEFFIC option (atmabs.dat, mirreff.dat, quanteff.dat) were provided by other CORSIKA users and I am probably not the right person to ask in case of problems with these files.*

Whether writing photon bunches or photo-electron bunches is more efficient in your case, is best determined by trying out both cases with appropriate bunch sizes. A properly designed telescope simulation program should be able to cope with both options. Photo-electron bunches (i.e. when the CEFFIC option is used) should be marked by a 'wavelength' parameter of -1, while a value of 0 indicates photon bunches (of undetermined wavelength within the given limits). Positive values, indicating a photon wavelength in nanometers are reserved for future enhancements.

c) For zenith angles above some 70 degrees, the CURVED option should be applied. In that case, the emission altitude of the photons is meant to be the altitude in the common sense, above sea level. Calculation of the amount of air traversed, e.g. for calculating atmospheric transmission, or the distance to the emission point no longer scales exactly with the secant of the zenith angle then.

d) The compact photon bunch output format, requiring 16 bytes per bunch, has several limitations which should probably be of little relevance for current or near-future telescope systems, but should be kept in mind.

   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,

   $$\sec(z) * R < 32.7 \text{ m}$$

   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. Starting with version 1.21 of this package, some tests at the beginning of each event were introduced which should catch most violation of the limits, although individual photon bunches still exceeding the limits cannot be fully ruled out. When any of these limits can be exceeded, the longer output format, requiring 32 bytes per bunch, should be used. This is achieved by prefixing the output file name (after the TELFIL keyword) with a '+', without any blanks inbetween, as e.g. in

```
TELFIL +iact.dat
```

   or

```
TELFIL +|analysis_program
```

   When reading data in compact format it is automatically converted to the longer format, bunch by bunch. Therefore, you need not bother with the internal representation of the data format.

e) If disk space is an important limitation, you may add a `.gz` or `.bz2` to your file names to force compression through the `gzip` or `bzip2` programs, respectively. Note that, due to the efficient eventio file format, compression rates are only of the order of 30%. If your analysis programs opens these files through the `fileopen()` function rather than the standard `fopen()` function, decompression on reading will be automatic.

f) When the IACT option is used together with the THIN option, all bunch sizes are simply multiplied by the weight, with no changes in the output format. Since the limit in d2) above is then easily violated, the compact bunch format should not be used together with the THIN option. Well, for imaging atmospheric Cherenkov telescopes, the thinning option might be not really appropriate at all and this final limitation, therefore, irrelevant.

g) You may pipe the IACT output data directly into a telescope (or other detector) simulation program by using a setting like

```
TELFIL |analysis_program
```

or

```
TELFIL +|analysis_program
```

in the CORSIKA inputs file. Note that due to restrictions of CORSIKA's processing of inputs files, you cannot add any command line option to your analysis program there. In case you need such options, use an intermediate shell script or similar to call your analysis program with options. The analysis program should read the data from its standard input then.

h) Note that the positions given on the 'TELESCOPE' lines in CORSIKA inputs file are in centimeters with respect to the nominal core position and the CORSIKA detection level. They are NOT counted from sea level.

i) Starting with version 1.21 of this package, the VOLUMEDET option of CORSIKA is now supported. While it is determined at the time of CMZ extraction for CORSIKA, it is fully dynamic for the IACT option and determined at run time. When CORSIKA is extracted/compiled without the VOLUMEDET option, all random shower core offsets (see the CSCAT configuration keyword in the CORSIKA User's Guide) are in a horizontal plane. Before version 1.21 of this package, this was always the case. If CORSIKA is now extracted/compiled with the VOLUMEDET option, the random core offsets are in the shower plane, defined as perpendicular to the shower axis.

j) Starting with version 1.25, the package has been prepared for importance sampling of core position offsets. This would mean that actual core offsets can be generated in a non-uniform distribution and can extend to different distances, depending on primary energy, primary type, zenith angle and so on. This package, however, does not provide a real implementation of importance sampling (other than for testing that the later stages of the processing properly get the weights for each event). If you do nothing about it, you will get uniformly distributed core offsets as before. If you plan to make use of importance sampling, you have to replace the file 'sampling.c' with an implementation of your choice.

k) Starting with version 1.34, the deflection of charged primary particles in the geomagnetic field is accounted for (with TSTART on). Thus the nominal core position is no longer on the extrapolation of the original velocity vector but approximately where the primary particle would intersect the detection level, assuming no interaction and no multiple scattering take place. This is useful for single muons in calibration events but also for low-energy electrons. This can be disabled in CORSIKA 6.5xx through a configuration line

```
IACT impact_correction off
```

Konrad Bernlöhr [April 2006]

```
Files included in this distribution:

   File              Size   Comments
   ------------     ------   -----------------------------------------
   Copyright           721   [Copyright notice]
   GNUmakefile       20840   [Makefile used with GNU make]
   README            31530   [this file as plain text]
   README.ps         74104   [this file as Postscript file]
   atmo.c            43496   [source code for tabulated atmospheres and refraction]
   atmo.h             2222   [header file in modules linking with atmo.c]
   atmprof1.dat       2749   [atmospheric profile: tropical]
   atmprof2.dat       2655   [atmospheric profile: midlatitude summer]
   atmprof3.dat       2691   [atmospheric profile: midlatitude winter]
   atmprof4.dat       2748   [atmospheric profile: subarctic summer]
   atmprof5.dat       2763   [atmospheric profile: subarctic winter]
   atmprof6.dat       2748   [atmospheric profile: U.S. standard 1976]
   atmprof9.dat       2803   [atmospheric profile: antarctic winter]
   cmz_extract       22515   [UNIX shell script for building CORSIKA with make]
   eventio.c        119698   [source code for basic 'eventio' functions]
   eventio_de.ps    170402   [description of basic eventio data format (in German)]
   eventio_en.ps    145973   [description of basic eventio data format (in English)]
   fileopen.c        10726   [open files found in one of several paths]
   fileopen.h          531   [header file in modules linking with fileopen.c]
   iact.c           109509   [source code for IACT interface to CORSIKA]
   iact3d.ps        427991   [plot illustrating photon bunch selection method]
   iact_refman.pdf ~0.9 MB   [automatically generated source code documentation]
   initial.h          9778   [required include file for system-specific things]
   io_basic.h        10603   [required include file for 'eventio']
   io_simtel.c       36890   [source code for higher-level IACT I/O functions]
   listio.c           3364   [source code for utility to list 'eventio' blocks]
   mc_tel.h           7171   [required include file for IACT eventio interface]
   sampling.c         2774   [dummy skeleton for importance sampling of core offsets]
   sampling.h          223   [interface definition for importance sampling]
   sim_skeleton.c    23716   [skeleton of a program reading the IACT eventio data]
   straux.c           3924   [auxilliary string functions]
   straux.h            480   [header file for straux.c]
   testio.c          28965   [source code for 'eventio' test program]
   trapfpe.c           263   [helps trapping floating point errors with g77/gcc]
   warning.c         17224   [required auxilliary source code for 'eventio']
   warning.h          1529   [required include file for 'eventio']
```

2 CORSIKA add-on package IACT/ATMO: Module Index

2.1 CORSIKA add-on package IACT/ATMO: Modules

Here is a list of all modules:

3 CORSIKA add-on package IACT/ATMO: Hierarchical Index

3.1 CORSIKA add-on package IACT/ATMO: Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 CORSIKA add-on package IACT/ATMO: Data Structure Index

4.1 CORSIKA add-on package IACT/ATMO: Data Structures

Here are the data structures with brief descriptions:

5 CORSIKA add-on package IACT/ATMO: File Index

5.1 CORSIKA add-on package IACT/ATMO: File List

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

6 CORSIKA add-on package IACT/ATMO: Module Documentation

6.1 The listio program

Functions

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

6.1.1 Function Documentation

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

The main function of the listio program.

6.2 The testio program

Data Structures

- struct **test_struct**

Typedefs

- typedef test_struct **TEST_DATA**

Functions

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

Compare elements of test data structures.

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

  Main function for I/O test program.

- int read_test1 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

  Read test data with single-element functions.

- int **read_test1** ()
- int read_test2 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

  Read test data with vector functions as far as possible.

- int **read_test2** ()
- int write_test1 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

  Write test data with single-element functions.

- int **write_test1** ()
- int write_test2 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

  Write test data with vector functions as far as possible.

- int **write_test2** ()

Variables

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

6.2.1 Function Documentation

6.2.1.1 int datacmp (TEST_DATA ∗ *data1*, TEST_DATA ∗ *data2*)

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.)

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

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

6.2.1.3 int read_test1 (TEST_DATA ∗ *data*, IO_BUFFER ∗ *iobuf*)

Parameters:

data Pointer to test data structure

iobuf Pointer to I/O buffer

Returns:

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

6.2.1.4 int read_test2 (TEST_DATA ∗ *data*, IO_BUFFER ∗ *iobuf*)

Parameters:

data Pointer to test data structure

iobuf Pointer to I/O buffer

Returns:

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

6.2.1.5 int write_test1 (TEST_DATA ∗ *data*, IO_BUFFER ∗ *iobuf*)

Parameters:

data Pointer to test data structure

iobuf Pointer to I/O buffer

Returns:

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

6.2.1.6 int write_test2 (TEST_DATA ∗ *data*, IO_BUFFER ∗ *iobuf*)

Parameters:

data Pointer to test data structure

iobuf Pointer to I/O buffer

Returns:

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

7 CORSIKA add-on package IACT/ATMO: Data Structure Documentation

7.1 _struct_IO_BUFFER Struct Reference

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

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

Data Fields

- int aux_count

  May be used for dedicated buffers.

- unsigned char ∗ buffer

  Pointer to allocated data space.

- long buflen

  Usable length of data space.

- int byte_order

  Set if block is not in internal byte order.

- BYTE ∗ data

  Position for next get.

- int data_pending

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

- FILE ∗ input_file

  For use of stream I/O for input.

- int input_fileno

  For use of read() function for input.

- int is_allocated

  Indicates if buffer is allocated by eventio.

- long item_length [MAX_IO_ITEM_LEVEL]

  Length of each level of items.

- int item_level

  Current level of nesting of items.

- long item_start_offset [MAX_IO_ITEM_LEVEL]

  Where the item starts in buffer.

- long max_length

  The maximum length for extending the buffer.

- long min_length

  The initial and minimum length of the buffer.

- FILE ∗ output_file

  For use of stream I/O for output.

- int output_fileno

  For use of write() function for output.

- long **r_remaining**
- int regular

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

- long sub_item_length [MAX_IO_ITEM_LEVEL]

  Length of its sub-items.

- int(∗ user_function)()

  For use of special type of I/O.

- long w_remaining

  Byte available for reading/writing.

7.1.1 Field Documentation

7.1.1.1 BYTE∗ _struct_IO_BUFFER::data

../put...

7.1.1.2 int _struct_IO_BUFFER::is_allocated

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

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

- io_basic.h

7.2 _struct_IO_ITEM_HEADER Struct Reference

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

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

Data Fields

- int can_search

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

- long ident

  Identity number.

- int level

  Tells how many levels deep we are nested now.

- unsigned long type

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

- unsigned version

The version number used for the block.

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

- io_basic.h

7.3 bunch Struct Reference

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

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

Data Fields

- float ctime

  Arrival time (ns).

- float **cx**
- float cy

  Direction cosines of photon direction.

- float lambda

  Wavelength in nanometers or 0.

- float photons

  Number of photons in bunch.

- float **x**
- float y

  Arrival position relative to telescope (cm).

- float zem

  Height of emission point above sea level (cm).

7.3.1 Detailed Description

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

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

- mc_tel.h

7.4 camera_electronics Struct Reference

Parameters of the electronics of a telescope.

Data Fields

- int simulated

  Is 1 if the signal simulation was done.

- int telescope

  Telescope sequence number.

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

- sim_skeleton.c

7.5 compact_bunch Struct Reference

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

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

Data Fields

- short ctime

  ctime∗10 (0.1ns) after subtracting offset

- short **cx**
- short cy

  cx,cy∗30000

- short lambda

  (nm) or 0

- short log_zem

  log10(zem)∗1000

- short photons

  ph∗100

- short **x**
- short y

  x,y∗10 (mm)

7.5.1 Detailed Description

And that has a number of limitations: 1) Bunch sizes must be less than 327. 2) photon impact points in a horizontal plane through the centre of each detector sphere must be less than 32.7 m from the detector centre in both x and y coordinates. Thus, sec(z) $*$ R $<$ 32.7 m is required, with 'z' being the zenith angle and 'R' the radius of the 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.

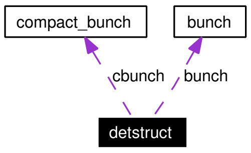
The documentation for this struct was generated from the following file:

- mc_tel.h

7.6 detstruct Struct Reference

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

Collaboration diagram for detstruct:



Data Fields

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

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

- iact.c

7.7 incpath Struct Reference

An element in a linked list of include paths.

Collaboration diagram for incpath:

Data Fields

- incpath ∗ next

  The next element.

- char ∗ path

  The path name.

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

- fileopen.c

7.8 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

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

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

- mc_tel.h

7.9 mc_run Struct Reference

Basic parameters of the CORSIKA run.

Data Fields

- double bunchsize

  Cherenkov bunch size.

- double e_max

  Upper limit of simulated energies [TeV].

- double e_min

  Lower limit of simulated energies [TeV].

- double height

  Height of observation level [m].

- int num_arrays

  Number of arrays simulated.

- double phi_max

  Upper limit of azimuth angle [degrees].

- double phi_min

  Lower limit of azimuth angle [degrees].

- double radius

  Radius within which cores are thrown at random.

- double slope

  Spectral index of power-law spectrum.

- double theta_max

  Upper limit of zenith angle [degrees].

- double theta_min

  Lower limit of zenith angle [degrees].

- double wlen_max

  Upper limit of Cherenkov wavelength range [nm].

- double wlen_min

  Lower limit of Cherenkov wavelength range [nm].

7.9.1 Field Documentation

7.9.1.1 double mc_run::radius

[m]

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

- sim_skeleton.c

7.10 photo_electron Struct Reference

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

`#include <mc_tel.h>`

Data Fields

- double atime

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

- int lambda

  The wavelength of the photon.

- int pixel

  The pixel that was hit.

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

- mc_tel.h

7.11 pm_camera Struct Reference

Parameters of a telescope camera (pixels, .

Data Fields

- int telescope

  Telescope sequence number.

7.11.1 Detailed Description

..)

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

- sim_skeleton.c

7.12 simulated_shower_parameters Struct Reference

Basic parameters of a simulated shower.

Data Fields

- double altitude

  Shower direction altitude above horizon.

- double azimuth

  Shower direction azimuth [deg].

- double cmax

  *Depth of maximum of Cherenkov light emission [g/cm**2].*

- double core_dist_3d

  Distance of core from reference point.

- double emax

  Depth of shower maximum from positrons and electrons.

- double energy

  Shower energy [TeV].

- double hmax

  Height of shower maximum (from xmax above) [m] a.s.l.

- int particle

  Primary particle type [CORSIKA code].

- double **tel_core_dist_3d** [MAX_TEL]
- double **xcore**
- double xmax

  *Depth of shower maximum from all particles [g/cm**2].*

- double **ycore**
- double zcore

  Shower core position [m].

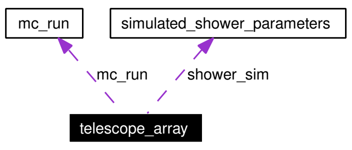
The documentation for this struct was generated from the following file:

- sim_skeleton.c

7.13 telescope_array Struct Reference

Description of telescope position, array offets and shower parameters.

Collaboration diagram for telescope_array:



Data Fields

- double altitude

  Nominal altitude angle of telescope system [deg].

- double azimuth

  Nominal azimuth angle of telescope system [deg].

- int max_tel

  Maximum number of telescopes acceptable (MAX_TEL).

- mc_run **mc_run**
- int narray

  Number of arrays with random shifts per shower.

- int ntel

  Number of telescopes simulated per array.

- double obs_height

  Height of observation level [cm].

- double refpos [3]

  Reference position with respect to obs.

- double rtel [MAX_TEL]

  Radius of spheres enclosing telescopes [cm].

- simulated_shower_parameters **shower_sim**
- double source_altitude

  Altitude of assumed source.

- double source_azimuth

  Azimuth of assumed source.

- double toff

  Time offset from first interaction to the moment when the extrapolated primary flying with the vacuum speed of light would be at the observation level.

- double xoff [MAX_ARRAY]

  X offsets of the randomly shifted arrays [cm].

- double xtel [MAX_TEL]

  X positions of telescopes ([cm] -> north).

- double yoff [MAX_ARRAY]

  Y offsets of the randomly shifted arrays [cm].

- double ytel [MAX_TEL]

  Y positions of telescopes ([cm] -> west).

- double ztel [MAX_TEL]

  Z positions of telescopes ([cm] -> up).

7.13.1 Field Documentation

7.13.1.1 double telescope_array::refpos[3]

level [cm]

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

- sim_skeleton.c

7.14 telescope_optics Struct Reference

Parameters describing the telescope optics.

Data Fields

- int telescope

  Telescope sequence number.

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

- sim_skeleton.c

7.15 warn_specific_data Struct Reference

A struct used to store thread-specific data.

Data Fields

- char ∗(∗ **aux_function**)()
- int **buffered**
- void(∗ **log_function**)()
- FILE ∗ **logfile**
- const char ∗ logfname

  The name of the log file.

- char **output_buffer** [2048]
- void(∗ **output_function**)()
- int **recursive**
- char **saved_logfname** [256]
- int **warninglevel**
- int **warningmode**

7.15.1 Field Documentation

7.15.1.1 const char∗ warn_specific_data::logfname

Used only when opening the file.

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

- warning.c

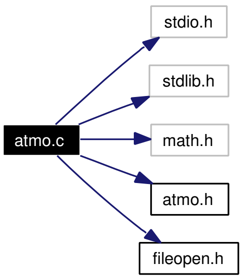
8 CORSIKA add-on package IACT/ATMO: File Documentation

8.1 atmo.c File Reference

Use of tabulated atmospheric profiles and atmospheric refraction.

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

Include dependency graph for atmo.c:



Defines

- #define **FAST_INTERPOLATION** 1
- #define **MAX_FAST_PROFILE** 10000
- #define **MAX_PROFILE** 50

Functions

- static double atm_exp_fit (double h1, double h2, double ∗ap, double ∗bp, double ∗cp, double ∗s0, int ∗npp)

  Fit one atmosphere layer by an expontential density model.

- void atmfit_ (int ∗nlp, double ∗hlay, double ∗aatm, double ∗batm, double ∗catm)

  Fit the tabulated density profile for CORSIKA EGS part.

- void atmset_ (int ∗iatmo, double ∗obslev)

  Set number of atmospheric model profile to be used.

- static double fn_rhof (double h, int nl, double ∗hl, double ∗a, double ∗b, double ∗c)

  Corresponding to CORSIKA built-in function RHOF; only used to show fit results.

- static double fn_thick (double h, int nl, double ∗hl, double ∗a, double ∗b, double ∗c)

  Corresponding to CORSIKA built-in function THICK; only used to show fit results.

- double heighx_ (double ∗thick)

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

- static void init_atmosphere ()

  Initialize atmospheric profiles.

- static void init_corsika_atmosphere ()

  Take the atmospheric profile from CORSIKA built-in functions.

- static void init_fast_interpolation ()

  An alternate interpolation method (which requires that the table is sufficiently fine-grained and equidistant) has to be initialized first.

- static void init_refraction_tables ()

  Initialize tables needed for atmospheric refraction.

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

  Linear interpolation with binary search algorithm.

- void raybnd_ (double ∗zem, cors_real_now_t ∗u, cors_real_now_t ∗v, double ∗w, cors_real_now_t ∗dx, cors_real_now_t ∗dy, cors_real_now_t ∗dt)

  Calculate the bending of light due to atmospheric refraction.

- double refidx_ (double ∗height)

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

- double rhofx_ (double ∗height)

  Density of the atmosphere as a function of altitude.

- double rpol (double ∗x, double ∗y, int n, double xp)

  Linear interpolation with binary search algorithm.

- static double sum_log_dev_sq (double a, double b, double c, int np, double ∗h, double ∗t, double ∗rho)

  Measure of deviation of model layers from tables.

- double thickx_ (double ∗height)

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

Variables

- int atmosphere

  The atmospheric profile number, 0 for built-in.

- static double **bottom_of_atmosphere** = 0.
- static double etadsn

  About the same as in CORSIKA Cherenkov function (but doesn't need to be the same).

- static double **fast_h_fac**
- static double **fast_p_alt** [MAX_FAST_PROFILE]
- static double **fast_p_log_n1** [MAX_FAST_PROFILE]
- static double **fast_p_log_rho** [MAX_FAST_PROFILE]
- static double **fast_p_log_thick** [MAX_FAST_PROFILE]
- static int **num_prof**
- static double **obs_level_refidx**
- static double **obs_level_thick**
- static double observation_level

  Altitude [cm] of observation level.

- static double **p_alt** [MAX_PROFILE]
- static double **p_bend_ray_hori_a** [MAX_PROFILE]
- static double **p_bend_ray_time0** [MAX_PROFILE]
- static double **p_bend_ray_time_a** [MAX_PROFILE]
- static double **p_log_alt** [MAX_PROFILE]
- static double **p_log_n1** [MAX_PROFILE]
- static double **p_log_rho** [MAX_PROFILE]
- static double **p_log_thick** [MAX_PROFILE]
- static double **p_rho** [MAX_PROFILE]
- static double **top_of_atmosphere** = 112.83e5

8.1.1 Detailed Description

Author:

Konrad Bernloehr

Date

2006/01/10 17:56:35

Revision

1.10

Copyright (C) 1990, 1997, 1998 Konrad Bernlohr. All rights reserved. Distribution and use of this software with the CORSIKA program is allowed and free. No redistribution separate of CORSIKA or of modified versions granted without permission. Modifications may, however, be distributed as patches to the original version. This software comes with no warranties.

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

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

8.1.2 Function Documentation

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

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

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

This function may be called from CORSIKA.

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

Parameters:

nlp Number of layers (or negative of that if boundaries set manually)

hlay Vector of layer (lower) boundaries.

aatm,batm,catm Parameters as used in CORSIKA.

8.1.2.2 void atmset_ (int ∗ *iatmo*, double ∗ *obslev*)

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

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

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

Parameters:

iatmo (pointer to) atmospheric profile number; negative for CORSIKA built-in profiles.

obslev (pointer to) altitude of observation level [cm]

Returns:

(none)

8.1.2.3 double heighx_ (double ∗ *thick*)

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

Parameters:

thick (pointer to) atmospheric thickness [g/cm∗∗2]

Returns:

altitude [cm]

8.1.2.4 static void init_atmosphere (void) `[static]`

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

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

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

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

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

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

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

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

8.1.2.8 void raybnd_ (double ∗ *zem*, cors_real_now_t ∗ *u*, cors_real_now_t ∗ *v*, double ∗ *w*, cors_real_now_t ∗ *dx*, cors_real_now_t ∗ *dy*, cors_real_now_t ∗ *dt*)

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

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

Parameters:

zem Altitude of emission above sea level [cm]

u Initial/Final direction cosine along X axis (updated)

v Initial/Final direction cosine along Y axis (updated)

w Initial/Final direction cosine along Z axis (updated)

dx Position in CORSIKA detection plane [cm] (updated)

dy Position in CORSIKA detection plane [cm] (updated)

dt Time of photon [ns]. Input: emission time. Output: time of arrival in CORSIKA detection plane.

8.1.2.9 double refidx_ (double ∗ *height*)

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

Parameters:

height (pointer to) altitude [cm]

Returns:

index of refraction

8.1.2.10 double rhofx_ (double ∗ *height*)

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

Parameters:

height (pointer to) altitude [cm]

Returns:

density [g/cm∗∗3]

8.1.2.11 double rpol (double ∗ *x*, double ∗ *y*, int *n*, double *xp*)

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value.

This 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

8.1.2.12 double thickx_ (double ∗ *height*)

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

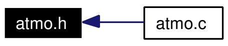
Parameters:
 height (pointer to) altitude [cm]

Returns:
 thickness [g/cm∗∗2]

8.2 atmo.h File Reference

Use of tabulated atmospheric profiles and atmospheric refraction.

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



Defines

- #define **ATMO_H__LOADED** 1
- #define **CORSIKA_VERSION** 6000

Typedefs

- typedef double **cors_real_now_t**

Functions

- void atmfit_ (int ∗nlp, double ∗hlay, double ∗aatm, double ∗batm, double ∗catm)

  Fit the tabulated density profile for CORSIKA EGS part.

- void atmset_ (int ∗iatmo, double ∗obslev)

  Set number of atmospheric model profile to be used.

- double heigh_ (double ∗thick)

  The CORSIKA built-in function for the height as a function of overburden.

- double heighx_ (double ∗thick)

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

- void raybnd_ (double ∗zem, cors_real_now_t ∗u, cors_real_now_t ∗v, double ∗w, cors_real_now_t ∗dx, cors_real_now_t ∗dy, cors_real_now_t ∗dt)

  Calculate the bending of light due to atmospheric refraction.

- double refidx_ (double ∗height)

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

- double rhof_ (double ∗height)

The CORSIKA built-in density lookup function.

- double rhofx_ (double ∗height)

  Density of the atmosphere as a function of altitude.

- double rpol (double ∗x, double ∗y, int n, double xp)

  Linear interpolation with binary search algorithm.

- double thick_ (double ∗height)

  The CORSIKA built-in function for vertical atmospheric thickness (overburden).

- double thickx_ (double ∗height)

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

8.2.1 Detailed Description

Author:

Konrad Bernloehr

Date

2005/06/08 18:02:31

Revision

1.2

Copyright (C) 2001 Konrad Bernloehr. All rights reserved. Distribution and use of this software with the CORSIKA program is allowed and free. No redistribution separate of CORSIKA or of modified versions granted without permission. Modifications may, however, be distributed as patches to the original version. This software comes with no warranties.

8.2.2 Function Documentation

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

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

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

This function may be called from CORSIKA.

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

Parameters:

nlp Number of layers (or negative of that if boundaries set manually)

hlay Vector of layer (lower) boundaries.

aatm,batm,catm Parameters as used in CORSIKA.

8.2.2.2 void atmset_ (int ∗ *iatmo*, double ∗ *obslev*)

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

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

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

Parameters:

iatmo (pointer to) atmospheric profile number; negative for CORSIKA built-in profiles.

obslev (pointer to) altitude of observation level [cm]

Returns:

(none)

8.2.2.3 double heighx_ (double ∗ *thick*)

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

Parameters:

thick (pointer to) atmospheric thickness [g/cm∗∗2]

Returns:

altitude [cm]

8.2.2.4 void raybnd_ (double ∗ *zem*, cors_real_now_t ∗ *u*, cors_real_now_t ∗ *v*, double ∗ *w*, cors_real_now_t ∗ *dx*, cors_real_now_t ∗ *dy*, cors_real_now_t ∗ *dt*)

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

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

Parameters:

zem Altitude of emission above sea level [cm]

u Initial/Final direction cosine along X axis (updated)

v Initial/Final direction cosine along Y axis (updated)

w Initial/Final direction cosine along Z axis (updated)

dx Position in CORSIKA detection plane [cm] (updated)

dy Position in CORSIKA detection plane [cm] (updated)

dt Time of photon [ns]. Input: emission time. Output: time of arrival in CORSIKA detection plane.

8.2.2.5 double refidx_ (double * *height*)

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

Parameters:

height (pointer to) altitude [cm]

Returns:

index of refraction

8.2.2.6 double rhofx_ (double * *height*)

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

Parameters:

height (pointer to) altitude [cm]

Returns:

density [g/cm**3]

8.2.2.7 double rpol (double * *x*, double * *y*, int *n*, double *xp*)

Linear interpolation between data point in sorted (i.e. monotonic ascending or descending) order. The resulting interpolated value is returned as a return value.

This 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

8.2.2.8 double thickx_ (double * *height*)

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

Parameters:

height (pointer to) altitude [cm]

Returns:

thickness [g/cm**2]

8.3 eventio.c File Reference

Basic functions for eventio data format.

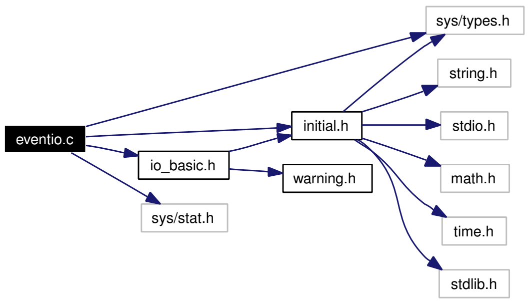
`#include "initial.h"`

`#include "io_basic.h"`

`#include <sys/types.h>`

`#include <sys/stat.h>`

Include dependency graph for eventio.c:



Defines

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

Functions

- IO_BUFFER ∗ allocate_io_buffer (size_t buflen)

  Dynamic allocation of an I/O buffer.

- int append_io_block_as_item (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header, BYTE ∗buffer, long length)

  Append data from one I/O block into another one.

- int copy_item_to_io_block (IO_BUFFER ∗iobuf2, IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Copy a sub-item to another I/O buffer as top-level item.

- int extend_io_buffer (IO_BUFFER ∗iobuf, unsigned next_byte, long increment)

  Extend the dynamically allocated I/O buffer.

- int find_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Find the beginning of the next I/O data block in the input.

- void free_io_buffer (IO_BUFFER ∗iobuf)

  Free an I/O buffer that has been allocated at run-time.

- uintmax_t get_count (IO_BUFFER ∗iobuf)

  Get an unsigned integer of unspecified length from an I/O buffer.

- uint16_t get_count16 (IO_BUFFER ∗iobuf)

  Get an unsigned 16 bit integer of unspecified length from an I/O buffer.

- double get_double (IO_BUFFER ∗iobuf)

  Get a double from the I/O buffer.

- int32_t get_int32 (IO_BUFFER ∗iobuf)

  Read a four byte integer from an I/O buffer.

- int get_item_begin (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Begin reading an item.

- int get_item_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  End reading an item.

- long get_long (IO_BUFFER ∗iobuf)

  Get 4-byte integer from I/O buffer and return as a long int.

- int get_long_string (char ∗s, int nmax, IO_BUFFER ∗iobuf)

  Get a long string of ASCII characters from an I/O buffer.

- double get_real (IO_BUFFER ∗iobuf)

  Get a floating point number (as written by put_real) from the I/O buffer.

- intmax_t get_scount (IO_BUFFER ∗iobuf)

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

- int16_t get_scount16 (IO_BUFFER ∗iobuf)

  Shortened version of get_scount for up to 16 bits of data.

- int get_short (IO_BUFFER ∗iobuf)

  Get a two-byte integer from an I/O buffer.

- int get_string (char ∗s, int nmax, IO_BUFFER ∗iobuf)

  Get a string of ASCII characters from an I/O buffer.

- uint32_t get_uint32 (IO_BUFFER ∗iobuf)

  Get a four-byte unsigned integer from an I/O buffer.

- int get_var_string (char ∗s, int nmax, IO_BUFFER ∗iobuf)

  Get a string of ASCII characters from an I/O buffer.

- void get_vector_of_byte (BYTE ∗vec, int num, IO_BUFFER ∗iobuf)

  Get a vector of bytes from an I/O buffer.

- void get_vector_of_double (double ∗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 get_vector_of_int (int ∗vec, int num, IO_BUFFER ∗iobuf)

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

  Get a vector of 32 bit integers from I/O buffer.
- void get_vector_of_long (long ∗vec, int num, IO_BUFFER ∗iobuf)

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

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

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

  Get a vector of unsigned shorts from an I/O buffer.
- void get_vector_of_uint32 (uint32_t ∗vec, int num, IO_BUFFER ∗iobuf)

  Get a vector of 32 bit integers from I/O buffer.
- int list_io_blocks (IO_BUFFER ∗iobuf)

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

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

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

  Reads the header of a sub-item and return the length of it.
- int next_subitem_type (IO_BUFFER ∗iobuf)

  Reads the header of a sub-item and return the type of it.
- void put_count (uintmax_t n, IO_BUFFER ∗iobuf)

  Put an unsigned integer of unspecified length to an I/O buffer.
- void put_count16 (uint16_t n, IO_BUFFER ∗iobuf)

  Shortened version of put_count for up to 16 bits of data.
- void put_double (double dnum, IO_BUFFER ∗iobuf)

  Put a 'double' as such into an I/O buffer.
- void put_int32 (int32_t num, IO_BUFFER ∗iobuf)

Write a four-byte integer to an I/O buffer.

- int put_item_begin (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Begin putting another (sub-) item into the output buffer.

- int put_item_end (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  End of putting an item into the output buffer.

- void put_long (long num, IO_BUFFER ∗iobuf)

  Put a four-byte integer taken from a 'long' into an I/O buffer.

- int put_long_string (char ∗s, IO_BUFFER ∗iobuf)

  Put a long string of ASCII characters into 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_scount (intmax_t n, IO_BUFFER ∗iobuf)

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

- void put_scount16 (int16_t n, IO_BUFFER ∗iobuf)

  Shorter version of put_scount for up to 16 bytes of data.

- void put_short (int num, IO_BUFFER ∗iobuf)

  Put a two-byte integer on an I/O buffer.

- int put_string (char ∗s, IO_BUFFER ∗iobuf)

  Put a string of ASCII characters into an I/O buffer.

- void put_uint32 (uint32_t num, IO_BUFFER ∗iobuf)

  Put a four-byte integer into an I/O buffer.

- int put_var_string (char ∗s, IO_BUFFER ∗iobuf)

  Put a string of ASCII characters into an I/O buffer.

- void put_vector_of_byte (BYTE ∗vec, int num, IO_BUFFER ∗iobuf)

  Put a vector of bytes into an I/O buffer.

- void put_vector_of_double (double ∗dvec, int num, IO_BUFFER ∗iobuf)

  Put a vector of doubles into an I/O buffer.

- void put_vector_of_float (float ∗fvec, int num, IO_BUFFER ∗iobuf)

  Put a vector of floats as IEEE 'float' numbers into an I/O buffer.

- void put_vector_of_int (int ∗vec, int num, IO_BUFFER ∗iobuf)

  Put a vector of integers (range -32768 to 32767) into I/O buffer.

- void put_vector_of_int32 (int32_t ∗vec, int num, IO_BUFFER ∗iobuf)

  Put a vector of 32 bit integers into I/O buffer.

- void put_vector_of_long (long ∗vec, int num, IO_BUFFER ∗iobuf)

  Put a vector of long int as 4-byte integers into an I/O buffer.

- void put_vector_of_real (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_short (short ∗vec, int num, IO_BUFFER ∗iobuf)

  Put a vector of 2-byte integers on an I/O buffer.

- void put_vector_of_uint16 (uint16_t ∗uval, int num, IO_BUFFER ∗iobuf)

  Put a vector of unsigned shorts into an I/O buffer.

- void put_vector_of_uint32 (uint32_t ∗vec, int num, IO_BUFFER ∗iobuf)

  Put a vector of 32 bit integers into I/O buffer.

- int read_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Read the data of an I/O block from the input.

- int remove_item (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Remove an item from an I/O buffer.

- int reset_io_block (IO_BUFFER ∗iobuf)

  Reset an I/O block to its empty status.

- int rewind_item (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Go back to the beginning of an item.

- int search_sub_item (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header, IO_ITEM_-HEADER ∗sub_item_header)

  Search for an item of a specified type.

- int skip_io_block (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Skip the data of an I/O block from the input.

- int skip_subitem (IO_BUFFER ∗iobuf)

  When the next sub-item is of no interest, it can be skipped.

- int unget_item (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Go back to the beginning of an item being read.

- int unput_item (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗item_header)

  Undo writing at the present level.

- int write_io_block (IO_BUFFER ∗iobuf)

  Write an I/O block to the block's output.

8.3.1 Detailed Description

Author:
Konrad Bernloehr

Date:
1991 to 2003

Date
2006/02/27 11:15:21

Revision
1.16

```
================ 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), reserved bits
     (16 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.
   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), reserved bits 16 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.

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'.

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.

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!

=====================================================================
```

8.3.2 Define Documentation

8.3.2.1 #define READ_BYTES(fd, buf, nb)

Value:

```
((fd==0) ? \
  fread((void *)buf,(size_t)1,(size_t)nb,stdin) : read(fd,buf,(size_t)nb))
```

8.3.3 Function Documentation

8.3.3.1 IO_BUFFER∗ allocate_io_buffer (size_t *buflen*)

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

Parameters:

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.

8.3.3.2 int append_io_block_as_item (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*, BYTE ∗ *buffer*, long *length*)

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

Parameters:

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 cuurently 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.)

8.3.3.3 int copy_item_to_io_block (IO_BUFFER ∗ *iobuf2*, IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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.)

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

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

Parameters:

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

8.3.3.5 int find_io_block (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

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)

8.3.3.6 void free_io_buffer (IO_BUFFER ∗ *iobuf*)

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

Parameters:

iobuf The buffer descriptor to be de-allocated.

Returns:

(none)

8.3.3.7 uintmax_t get_count (IO_BUFFER ∗ *iobuf*)

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

8.3.3.8 uint16_t get_count16 (IO_BUFFER ∗ *iobuf*)

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

8.3.3.9 double get_double (IO_BUFFER ∗ *iobuf*)

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

Parameters:

iobuf – The I/O buffer descriptor;

Returns:

The floating point number.

8.3.3.10 int32_t get_int32 (IO_BUFFER ∗ *iobuf*)

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

8.3.3.11 int get_item_begin (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

Reads the header of an item.

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

Parameters:

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).

8.3.3.12 int get_item_end (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf I/O buffer descriptor.

item_header Header of item last read.

Returns:

0 (ok), -1 (error)

8.3.3.13 long get_long (IO_BUFFER ∗ *iobuf*)

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

8.3.3.14 int get_long_string (char ∗ *s*, int *nmax*, IO_BUFFER ∗ *iobuf*)

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

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

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

8.3.3.15 double get_real (IO_BUFFER ∗ *iobuf*)

Parameters:

iobuf The I/O buffer descriptor;

Returns:

The floating point number.

8.3.3.16 intmax_t get_scount (IO_BUFFER ∗ *iobuf*)

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

8.3.3.17 int get_short (IO_BUFFER ∗ *iobuf*)

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

8.3.3.18 int get_string (char ∗ *s*, int *nmax*, IO_BUFFER ∗ *iobuf*)

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

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

8.3.3.19 uint32_t get_uint32 (IO_BUFFER ∗ *iobuf*)

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

8.3.3.20 int get_var_string (char ∗ *s*, int *nmax*, IO_BUFFER ∗ *iobuf*)

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

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

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

Parameters:

vec – Byte data vector.

num – Number of bytes to get.

iobuf – I/O buffer descriptor.

Returns:

(none)

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

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

Parameters:

uval The vector where the values should be loaded.

num The number of elements to load.

iobuf The output buffer descriptor.

Returns:

(none)

8.3.3.23 int list_io_blocks (IO_BUFFER ∗ *iobuf*)

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.

Returns:

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

8.3.3.24 int list_sub_items (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*, int *maxlevel*)

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).

Returns:

0 (ok), -1 (error)

8.3.3.25 long next_subitem_ident (IO_BUFFER ∗ *iobuf*)

Parameters:

iobuf The input buffer descriptor.

Returns:

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

8.3.3.26 long next_subitem_length (IO_BUFFER ∗ *iobuf*)

Parameters:

iobuf The input buffer descriptor.

Returns:

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

8.3.3.27 int next_subitem_type (IO_BUFFER ∗ *iobuf*)

Parameters:

iobuf The input buffer descriptor.

Returns:

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

8.3.3.28 void put_count (uintmax_t *n*, IO_BUFFER ∗ *iobuf*)

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

Parameters:

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)

8.3.3.29 void put_count16 (uint16_t *n*, IO_BUFFER ∗ *iobuf*)

Returns:

(none)

8.3.3.30 void put_double (double *dnum*, IO_BUFFER ∗ *iobuf*)

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

Parameters:

dnum The number to be put into the I/O buffer.

iobuf The I/O buffer descriptor.

Returns:

(none)

8.3.3.31 void put_int32 (int32_t *num*, IO_BUFFER ∗ *iobuf*)

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

8.3.3.32 int put_item_begin (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

When putting another item to the output buffer which may be either a top item or a sub-item, put_item_-begin() initializes the buffer (for a top item) and puts the item header on the buffer.

Parameters:

iobuf The output buffer descriptor.

item_header The item header descriptor.

Returns:

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

8.3.3.33 int put_item_end (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf The output buffer descriptor.

item_header The item header descriptor.

Returns:

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

8.3.3.34 void put_long (long *num*, IO_BUFFER ∗ *iobuf*)

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

8.3.3.35 int put_long_string (char * *s*, IO_BUFFER * *iobuf*)

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

Parameters:

s The null-terminated ASCII string.

iobuf The I/O buffer descriptor.

Returns:

Length of string

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

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

Parameters:

dnum The number to be put into the I/O buffer.

iobuf The I/O buffer descriptor.

Returns:

(none)

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

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

Parameters:

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)

8.3.3.38 void put_scount16 (int16_t *n*, IO_BUFFER ∗ *iobuf*)

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

Returns:
(none)

8.3.3.39 void put_short (int *num*, IO_BUFFER ∗ *iobuf*)

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

8.3.3.40 int put_string (char ∗ *s*, IO_BUFFER ∗ *iobuf*)

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

Parameters:
s The null-terminated ASCII string.

iobuf The I/O buffer descriptor.

Returns:
Length of string

8.3.3.41 void put_uint32 (uint32_t *num*, IO_BUFFER ∗ *iobuf*)

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

8.3.3.42 int put_var_string (char ∗ *s*, IO_BUFFER ∗ *iobuf*)

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

Parameters:
s The null-terminated ASCII string.

iobuf The I/O buffer descriptor.

Returns:
Length of string

8.3.3.43 void put_vector_of_byte (BYTE ∗ *vec*, int *num*, IO_BUFFER ∗ *iobuf*)

Parameters:

vec Byte data vector.

num Number of bytes to be put.

iobuf I/O buffer descriptor.

Returns:

(none)

8.3.3.44 void put_vector_of_double (double ∗ *dvec*, int *num*, IO_BUFFER ∗ *iobuf*)

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

8.3.3.45 void put_vector_of_int (int ∗ *vec*, int *num*, IO_BUFFER ∗ *iobuf*)

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

8.3.3.46 void put_vector_of_short (short ∗ *vec*, int *num*, IO_BUFFER ∗ *iobuf*)

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.

8.3.3.47 void put_vector_of_uint16 (uint16_t ∗ *uval*, int *num*, IO_BUFFER ∗ *iobuf*)

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

Parameters:

uval The vector of values to be saved.

num The number of elements to save.

iobuf The output buffer descriptor.

Returns:

(none)

8.3.3.48 int read_io_block (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

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)

8.3.3.49 int remove_item (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf I/O buffer descriptor.

item_header Header of item to be removed.

Returns:

0 (ok), -1 (error)

8.3.3.50 int reset_io_block (IO_BUFFER ∗ *iobuf*)

Parameters:

iobuf The I/O buffer descriptor.

Returns:

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

8.3.3.51 int rewind_item (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf I/O buffer descriptor.

item_header Header of item last read.

Returns:

0 (ok), -1 (error)

8.3.3.52 int search_sub_item (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*, IO_ITEM_HEADER ∗ *sub_item_header*)

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

Parameters:

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),

8.3.3.53 int skip_io_block (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf The I/O buffer descriptor.

item_header The item header descriptor.

Returns:

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

8.3.3.54 int skip_subitem (IO_BUFFER ∗ *iobuf*)

Parameters:

iobuf I/O buffer descriptor.

Returns:

0 (ok), -1 (error)

8.3.3.55 int unget_item (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf I/O buffer descriptor.

item_header Header of item last read.

Returns:

0 (ok), -1 (error)

8.3.3.56 int unput_item (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *item_header*)

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

Parameters:

iobuf I/O buffer descriptor.

item_header Header of item last read.

Returns:

0 (ok), -1 (error)

8.3.3.57 int write_io_block (IO_BUFFER ∗ *iobuf*)

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

Parameters:

iobuf The I/O buffer descriptor.

Returns:

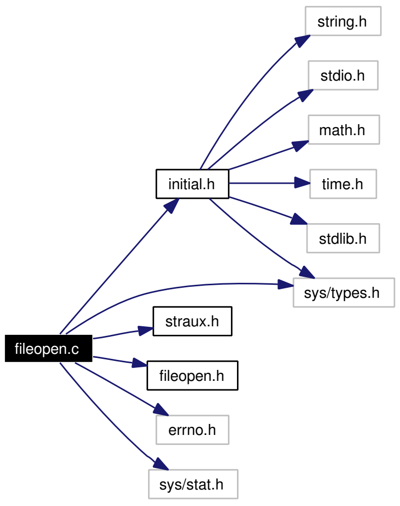
0 (O.k.), -1 (error), -2 (item has no data)

8.4 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 addpath (const char ∗name)

Add a path to the list of include paths, if not already there.

- static FILE ∗ cmp_popen (const char ∗fname, const char ∗mode, int compression)

  Helper function for opening a compressed file through a fifo.

- int fileclose (FILE ∗f)

  Close a file or fifo but not if it is one of the standard streams.

- FILE ∗ fileopen (const char ∗fname, const char ∗mode)

  Search for a file in the include path list and open it if possible.

- static void freepath ()

  Free a whole list of include path elements.

- void initpath (const char ∗default_path)

  Init the path list, with default_path as the only entry.

- void listpath (char ∗buffer, int bufsize)

  Show the list of include paths.

Variables

- static struct incpath ∗ root_path = NULL

  The starting element of include paths.

8.4.1 Detailed Description

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

- Where possible files are opened such that more than 2 gigabytes of data can be accessed on 32-bit systems when suitably compiled. This also works with software where a '-D_FILE_OFFSET_-BITS=64' at compile-time cannot be used (of which ROOT is an infamous example).
- For reading files, a list of paths can be configured before the the first fileopen() call and all files without absolute paths will be searched in these paths. Writing always strictly follows the given file name and will not search in the path list.
- Files compressed with `gzip` or `bzip2` can be handled on the fly. Files with corresponding file name extensions will be automatically decompressed when reading or compressed when writing (in a pipe, i.e. without producing temporary copies).

Author:

Konrad Bernloehr

Date:

Nov. 2000
CVS
Date
2003/09/12 21:09:43

Version:

CVS

Revision

1.2

8.4.2 Function Documentation

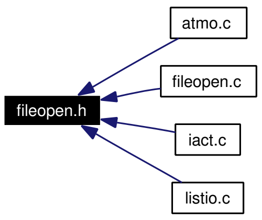
8.4.2.1 void addpath (const char ∗ *name*)

The path name is always copied to a newly allocated memroy location.

8.5 fileopen.h File Reference

Function prototypes for fileopen.c.

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



Defines

- #define **FILEOPEN_H__LOADED** 1

Functions

- void addpath (const char ∗name)

  Add a path to the list of include paths, if not already there.

- int fileclose (FILE ∗f)

  Close a file or fifo but not if it is one of the standard streams.

- FILE ∗ fileopen (const char ∗fname, const char ∗mode)

  Search for a file in the include path list and open it if possible.

- void initpath (const char ∗default_path)

  Init the path list, with default_path as the only entry.

- void listpath (char ∗buffer, int bufsize)

  Show the list of include paths.

8.5.1 Detailed Description

Author:
Konrad Bernloehr

Date:
CVS
Date
2003/04/30 18:10:12

Version:
CVS
Revision
1.3

8.5.2 Function Documentation

8.5.2.1 void addpath (const char ∗ *name*)

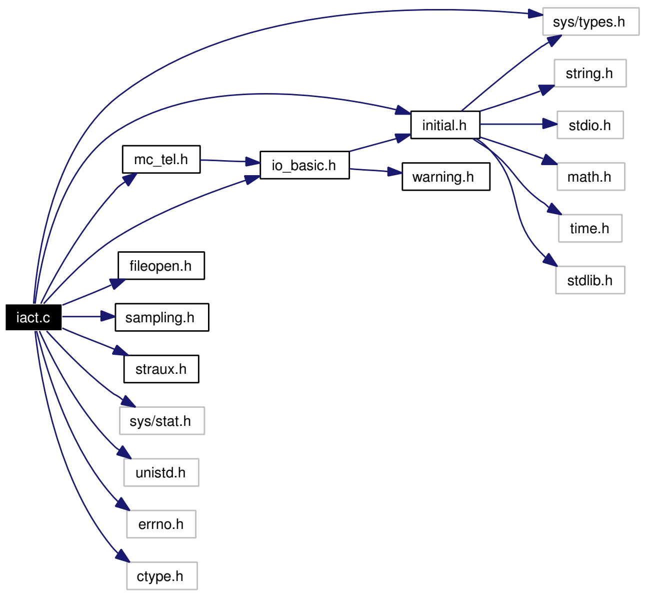
The path name is always copied to a newly allocated memroy location.

8.6 iact.c File Reference

CORSIKA interface for Imaging Atmospheric Cherenkov Telescopes etc.

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

Include dependency graph for iact.c:



Data Structures

- struct detstruct

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

- struct **gridstruct**

Defines

- #define **CORSIKA_VERSION** 6500
- #define EXTERNAL_STORAGE 1

  Enable external temporary bunch storage.

- #define **EXTRA_MEM** 0
- #define **EXTRA_MEM_1** EXTRA_MEM
- #define **EXTRA_MEM_10** EXTRA_MEM
- #define **EXTRA_MEM_11** EXTRA_MEM
- #define **EXTRA_MEM_12** EXTRA_MEM
- #define **EXTRA_MEM_2** EXTRA_MEM
- #define **EXTRA_MEM_3** EXTRA_MEM
- #define **EXTRA_MEM_4** EXTRA_MEM
- #define **EXTRA_MEM_5** EXTRA_MEM
- #define **EXTRA_MEM_6** EXTRA_MEM
- #define **EXTRA_MEM_7** EXTRA_MEM
- #define **EXTRA_MEM_8** EXTRA_MEM
- #define **EXTRA_MEM_9** EXTRA_MEM
- #define GRID_SIZE 1000

  unit: cm

- #define **HAVE_EVENTIO_FUNCTIONS** 1

- #define **IACT_ATMEXT_VERSION** "1.35 (2006-04-06)"
- #define INTERNAL_LIMIT 100000

  Start external storage after so many bunches.

- #define **max**(a, b) ((a)>(b)?(a):(b))
- #define MAX_ARRAY_SIZE 1000

  Maximum number of telescopes (or other detectors) per array.

- #define **MAX_CLASS** 1
- #define **MAX_IO_BUFFER** 200000000
- #define **min**(a, b) ((a)<(b)?(a):(b))
- #define NBUNCH 1000

  Memory allocation step size for bunches.

- #define **PIPE_OUTPUT** 1
- #define **PRMPAR_SIZE** 17
- #define **square**(x) ((x)∗(x))
- #define **UNKNOWN_LONG_DIST** 1

Typedefs

- typedef double cors_dbl_t

  Type for CORSIKA numbers which were already REAL∗8.

- typedef double cors_real_dbl_t

  Type for CORSIKA numbers which were REAL∗4 but changed to REAL∗8 at 5.900.

- typedef double cors_real_now_t

  Type for many CORSIKA numbers has changed to REAL∗8 with version 5.901.

- typedef float cors_real_t

  Type for CORSIKA floating point numbers remaining REAL∗4.

Functions

- static int compact_photon_hit (struct detstruct ∗det, double x, double y, double cx, double cy, double sx, double sy, double photons, double ctime, double zem, double lambda)

  Store a photon bunch for a given telescope in compact format.

- void get_impact_offset (cors_real_t evth[273], cors_real_dbl_t prmpar[PRMPAR_SIZE])

  Approximate impact offset of primary due to geomagnetic field.

- double **heigh_** (double ∗thickness)
- static void iact_param (char ∗text)

  Processing of IACT module specific parameters in Corsika input.

- double **iact_rndm** (int dummy)
- static int in_detector (struct detstruct ∗det, double x, double y, double sx, double sy)

Check if a photon bunch hits a particular telescope volume.

- static void **ioerrorcheck** ()
- static int **is_off** (char ∗word)
- static int **is_on** (char ∗word)
- static int Nint_f (double x)

  Nearest integer function.

- static int photon_hit (struct detstruct ∗det, double x, double y, double cx, double cy, double sx, double sy, double photons, double ctime, double zem, double lambda)

  Store a photon bunch for a given telescope in long format.

- double refidx_ (double ∗height)

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

- double **rhof_** (double ∗height)
- void **rmmard_** (double ∗, int ∗, int ∗)
- static double rndm (int dummy)

  Random number interface using sequence 4 of CORSIKA.

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

  Get uniformly sampled or importance sampled offset of array with respect to core, in the plane perpendicular to the shower axis.

- static int set_random_systems (double theta, double phi, double thetaref, double phiref, double offax, double E, int primary, int volflag)

  Randomly scatter each array of detectors in given area.

- void telasu_ (int ∗n, cors_real_dbl_t ∗dx, cors_real_dbl_t ∗dy)

  Setup how many times each shower is used.

- void telend_ (cors_real_t evte[273])

  End of event.

- void televt_ (cors_real_t evth[273], cors_real_dbl_t prmpar[PRMPAR_SIZE])

  Start of new event.

- void telfil_ (char ∗name)

  Define the output file for photon bunches hitting the telescopes.

- void telinf_ (int ∗itel, double ∗x, double ∗y, double ∗z, double ∗r, int ∗exists)

  Return information about configured telescopes back to CORSIKA.

- void tellng_ (int ∗type, double ∗data, int ∗ndim, int ∗np, int ∗nthick, double ∗thickstep)

  Write CORSIKA 'longitudinal' (vertical) distributions.

- void tellni_ (char ∗line, int ∗llength)

  Keep a record of CORSIKA input lines.

- int telout_ (cors_real_now_t ∗bsize, cors_real_now_t ∗wt, cors_real_now_t ∗px, cors_real_now_t ∗py, cors_real_now_t ∗pu, cors_real_now_t ∗pv, cors_real_now_t ∗ctime, cors_real_now_t ∗zem, cors_real_now_t ∗lambda)

  Check if a photon bunch hits one or more simulated detector volumes.

- void telrne_ (cors_real_t rune[273])

  Write run end block to the output file.

- void telrnh_ (cors_real_t runh[273])

  Save aparameters from CORSIKA run header.

- void telset_ (cors_real_now_t ∗x, cors_real_now_t ∗y, cors_real_now_t ∗z, cors_real_now_t ∗r)

  Add another telescope to the system (array) of telescopes.

- void telshw_ ()

  Show what telescopes have actually been set up.

- void telsmp_ (char ∗name)

  Set the file name with parameters for importance sampling.

Variables

- static double airlightspeed = 29.9792458/1.0002256

  [cm/ns] at H=2200 m

- static double **all_bunches**
- static double **all_bunches_run**
- static double **all_photons**
- static double **all_photons_run**
- static double core_range

  The maximum core offset of array centres in circular distribution.

- static double core_range1

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

- static double **core_range2**
- static struct linked_string **corsika_inputs** = { "∗ CORSIKA inputs:", NULL }
- int corsika_version = (CORSIKA_VERSION)

  The CORSIKA version actually running.

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

  Max.

- static int **do_print**

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

    Correct impact position if non-zero.

- static double impact_offset [2]

    Offset of impact position of charged primaries.

- static IO_BUFFER ∗ **iobuf**
- static double **lambda1**
- static double **lambda2**
- static int max_internal_bunches = INTERNAL_LIMIT

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

- static size_t max_io_buffer = MAX_IO_BUFFER

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

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

    Number of arrays.

- static int ntel = 0

    Number of telescopes set up.

- static double **obs_height**
- static double **off_axis**
- static char ∗ output_fname

    The name of the output file for eventio format data.

- static double **phi_central**
- static double **phi_prim**
- static int **primary**
- static double raise_tel

    Non-zero if any telescope has negative z.

- static double rmax = 0.

Max.

- static double **rtel** [MAX_ARRAY_SIZE]
- static char ∗ sampling_fname

  The name of the file providing parameters for importance sampling.

- static int **skip_off2** = 1
- static int **skip_print** = 1
- static int **skip_print2** = 100
- static long **stored_bunches**
- static int **televt_done**
- static double theta_central

  The central value of the allowed ranges in theta and phi.

- static double **theta_prim**
- static double **toffset**
- static int **use_compact_format** = 1
- static double **ush**
- static double **ushc**
- static double **vsh**
- static double **vshc**
- static double ∗ **weight**
- static double **wsh**
- static double **wshc**
- static double ∗ **xoffset**
- static double xtel [MAX_ARRAY_SIZE]

  Position and size definition of fiducial spheres.

- static double ∗ **yoffset**
- static double **ytel** [MAX_ARRAY_SIZE]
- static double **ztel** [MAX_ARRAY_SIZE]

8.6.1 Detailed Description

Author:

Konrad Bernloehr

Date

2006/04/07 11:38:39

Revision

1.43

Copyright (C) 1997, 1998, 1999, 2001, 2002, 2005, 2006 Konrad Bernloehr. All rights reserved. Distribution and use of this software with the CORSIKA program is allowed and free. No redistribution separate of CORSIKA or of modified versions granted without permission. Modifications may, however, be distributed as patches to the original version. This software comes with no warranty.

Version 1.2.9

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

General comments on this file:

Routines provided in this file interface to recent versions of the CORSIKA air shower simulation program. Modifications to CORSIKA have been kept as simple as possible and the existing routines for production of Cherenkov light have been largely maintained. Setup of the telescope systems to be simulated is via the usual CORSIKA input file (the syntax of which has been extended by a few additional keywords). These telescope systems can be randomly scattered several times within a given area. All treatment whether a bunch of photons hits a telescope is done by the routines in this file. Photon bunches are kept in main memory until the end of the event. This might be a limitation when simulating large showers / many telescopes / many systems of telescopes on a computer with little memory. An option to store photon bunches in a temporary file has, therefore, been included. After the end of an event in CORSIKA all photon bunches (sorted by system and telescope) are written to a data file in the 'eventio' portable data format also used for CRT and HEGRA CT data. All CORSIKA run/event header/trailer blocks are also written to this file.

8.6.2 Function Documentation

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

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

Parameters:

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

Returns:

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

8.6.2.2 void get_impact_offset (cors_real_t *evth*[273], cors_real_dbl_t *prmpar*[PRMPAR_SIZE])

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

Parameters:

evth CORSIKA event header block

prmpar CORSIKA primary particle block. We need it to get the particle's relativistic gamma factor (prmpar[2] or prmpar[1], depending on the CORSIKA version).

Returns:

(none)

8.6.2.3 static void iact_param (char ∗ *text*) `[static]`

Parameters:

text Text following the IACT keyword on the input line.

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

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

Parameters:

x X position of photon position in CORSIKA detection level [cm]

y Y position of photon position in CORSIKA detection level [cm]

sx Slope of photon direction in X/Z plane.

sy Slope of photon direction in Y/Z plane.

Returns:

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

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

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

Parameters:

det pointer to data structure of the detector hit.

x X position in CORSIKA detection plane [cm]

y Y position in CORSIKA detection plane [cm]

cx Direction projection onto X axis

cy Direction projection onto Y axis

sx Slope with respect to X axis (atan(sx) = acos(cx))

sy Slope with respect to Y axis (atan(sy) = acos(cy))

photons Bunch size

ctime Arrival time of bunch in CORSIKA detection plane.

zem Altitude of emission above sea level [cm]

lambda Wavelength (0: undetermined, -1: converted to photo-electron)

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

8.6.2.6 double refidx_ (double ∗ *height*)

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

Parameters:

height (pointer to) altitude [cm]

Returns:
index of refraction

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

Parameters:

sampling_fname Name of file with parameters, to be read on first call.

core_range Maximum core distance as used in data format check [cm]. If not obeying this maximum distance, make sure to switch on the long data format manually.

theta Zenith angle [radians]

phi Shower azimuth angle in CORSIKA angle convention [radians].

thetaref Reference zenith angle (e.g. of VIEWCONE centre) [radians].

phiref Reference azimuth angle (e.g. of VIEWCONE centre) [radians].

offax Angle between central direction (typically VIEWCONE centre) and the direction of the current primary [radians].

E Energy of primary particle [GeV]

primary Primary particle ID.

xoff X offset [cm] to be generated.

yoff Y offset [cm] to be generated.

sampling_area Area weight of the generated sample (normalized to Pi∗core_range^2) [cm^2].

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

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

Parameters:

theta Zenith angle of the shower following [radians].

phi Shower azimuth angle in CORSIKA angle convention [radians].

thetaref Reference zenith angle (e.g. of VIEWCONE centre) [radians].

phiref Reference azimuth angle (e.g. of VIEWCONE centre) [radians].

offax Angle between central direction (typically VIEWCONE centre) and the direction of the current primary [radians].

E Primary particle energy in GeV (may be used in importance sampling).

primary Primary particle ID (may be used in importance sampling).

volflag Set to 1 if CORSIKA was compiled with VOLUMEDET option, 0 otherwise.

Returns:

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

8.6.2.9 void telasu_ (int ∗ *n*, cors_real_dbl_t ∗ *dx*, cors_real_dbl_t ∗ *dy*)

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

Parameters:

n The number of telescope systems

dx Core range radius (if dy==0) or core x range

dy Core y range (non-zero for ractangular, 0 for circular)

Returns:

(none)

8.6.2.10 void telend_ (cors_real_t *evte*[273])

Write out all recorded photon bunches.

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

Parameters:

evte CORSIKA event end block

Returns:

(none)

8.6.2.11 void televt_ (cors_real_t *evth*[273], cors_real_dbl_t *prmpar*[PRMPAR_SIZE])

Save event parameters.

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

Parameters:

evth CORSIKA event header block

prmpar CORSIKA primary particle block

Returns:

(none)

8.6.2.12 void telfil_ (char ∗ *name*)

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

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

Parameters:

name Output file name. Note: A leading '+' means: use non-compact format A leading '|' (perhaps after '+') means that the name will not be interpreted as the name of a data file but of a program to which the 'eventio' data stream will be piped (i.e. that program should read the data from its standard input.

Returns:

(none)

8.6.2.13 void telinf_ (int ∗ *itel*, double ∗ *x*, double ∗ *y*, double ∗ *z*, double ∗ *r*, int ∗ *exists*)

Parameters:

itel number of telescope in question

x,y,z telescope position [cm]

r radius of fiducial volume [cm]

exists telescope exists

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

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

```
 *  There are three types of distributions:
 * type 1: particle distributions for
 * gammas, positrons, electrons, mu+, mu-,
 * hadrons, all charged, nuclei, Cherenkov photons.
 * type 2: energy distributions (with energies in GeV) for
 * gammas, positrons, electrons, mu+, mu-,
 * hadrons, all charged, nuclei, sum of all.
 * type 3: energy deposits (in GeV) for
 * gammas, e.m. ionisation, cut of e.m.  particles,
 * muon ionisation, muon cut, hadron ionisation,
 * hadron cut, neutrinos, sum of all.
 * ('cut' accounting for low-energy particles dropped)
 *
```

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

Parameters:

type see above

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:

(none)

8.6.2.15 void tellni_ (char ∗ *line*, int ∗ *llength*)

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

Parameters:

line input line (not terminated)

llength maximum length of input lines (132 usually)

8.6.2.16 int telout_ (cors_real_now_t ∗ *bsize*, cors_real_now_t ∗ *wt*, cors_real_now_t ∗ *px*, cors_-real_now_t ∗ *py*, cors_real_now_t ∗ *pu*, cors_real_now_t ∗ *pv*, cors_real_now_t ∗ *ctime*, cors_real_-now_t ∗ *zem*, cors_real_now_t ∗ *lambda*)

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

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

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

Parameters:

bsize Number of photons (can be fraction of one)

wt Weight (if thinning option is active)

px x position in detection level plane

py y position in detection level plane

pu x direction cosine

pv y direction cosine

ctime arrival time in plane after first interaction

zem height of emission above sea level

lambda 0. (if wavelength undetermined) or wavelength [nm]. If lambda < 0, photons are already converted to photo-electrons (p.e.), i.e. we have p.e. bunches.

temis Time of photon emission (only if CORSIKA extracted with IACTEXT option and this code compiled with EXTENDED_TELOUT defined).

penergy Energy of emitting particle (under conditions as temis).

amass Mass of emitting particle (under conditions as temis).

charge Charge of emitting particle (under conditions as temis).

Returns:

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

8.6.2.17 void telrne_ (cors_real_t *rune*[273])

Parameters:

rune CORSIKA run end block

8.6.2.18 void telrnh_ (cors_real_t *runh*[273])

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

Parameters:

runh CORSIKA run header block

Returns:

(none)

8.6.2.19 void telset_ (cors_real_now_t ∗ *x*, cors_real_now_t ∗ *y*, cors_real_now_t ∗ *z*, cors_real_-now_t ∗ *r*)

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

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

Parameters:

x X position [cm]

y Y position [cm]

z Z position [cm]

r radius [cm] within which the telescope is fully contained

Returns:

(none)

8.6.2.20 void telshw_ (void)

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

8.6.2.21 void telsmp_ (char ∗ *name*)

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

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

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

8.6.3 Variable Documentation

8.6.3.1 double dmax = 0. `[static]`

distance of telescopes in (x,y)

8.6.3.2 double rmax = 0. `[static]`

radius of telescopes

8.7 initial.h File Reference

Indentification of the system and including some basic include file.

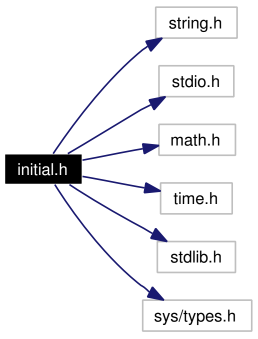
`#include <string.h>`

`#include <stdio.h>`

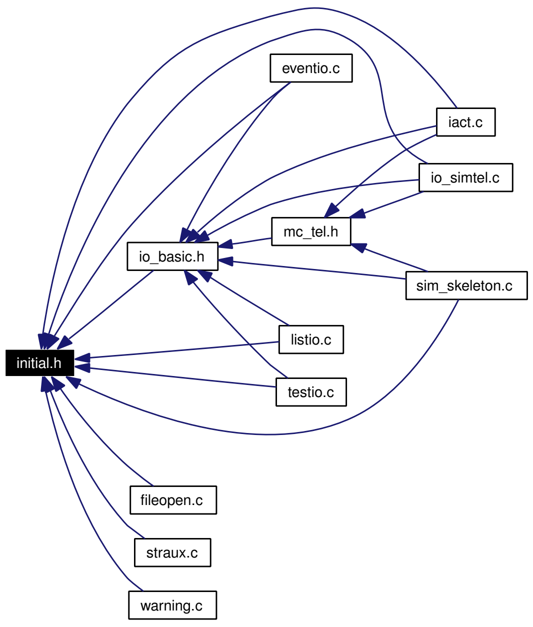
`#include <math.h>`

```
#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:



Defines

- #define **Abs**(a) (((a)>=0)?(a):(-1∗(a)))
- #define **APPEND_BINARY** "a"
- #define **APPEND_TEXT** "a"
- #define **ARGLIST**(a) a
- #define **CONST_QUAL**
- #define **IEEE_FLOAT_FORMAT** 1

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

Typedefs

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

8.7.1 Detailed Description

Author:
Konrad Bernloehr

Date:
1991 to 2000

Date
2006/02/27 11:15:21

Version: **Revision**
1.7

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
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_VAX
   CPU_MIPS
   CPU_ALPHA
   CPU_68K
   CPU_RS6000
   CPU_PowerPC
   CPU_HPPA
```

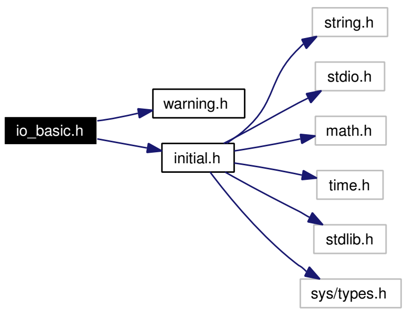
8.8 io_basic.h File Reference

Basic header file for eventio data format.

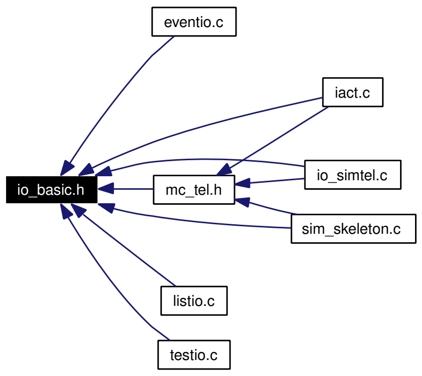
`#include "warning.h"`

`#include "initial.h"`

Include dependency graph for io_basic.h:



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



Data Structures

- struct _struct_IO_BUFFER

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

- struct _struct_IO_ITEM_HEADER

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

Defines

- #define **COPY_BYTES**(_target, _source, _num) memcpy(_target,_source,_num)
- #define **get_byte**(p) (–(p) → r_remaining>=0? ∗(p) → data++ : -1)
- #define **get_vector_of_int16** get_vector_of_short
- #define **get_vector_of_uint8** get_vector_of_byte
- #define **IO_BASIC_H__LOADED** 1
- #define **IO_BUFFER_INITIAL_LENGTH** 32768L
- #define **IO_BUFFER_LENGTH_INCREMENT** 65536L
- #define **IO_BUFFER_MAXIMUM_LENGTH** 3000000L
- #define **MAX_IO_ITEM_LEVEL** 20
- #define **put_byte**(_c, _p)
- #define **put_vector_of_int16** put_vector_of_short
- #define **put_vector_of_uint8** put_vector_of_byte

Typedefs

- typedef unsigned char **BYTE**
- typedef _struct_IO_BUFFER **IO_BUFFER**
- typedef _struct_IO_ITEM_HEADER **IO_ITEM_HEADER**
- typedef int(∗ **IO_USER_FUNCTION**)()

Functions

- IO_BUFFER ∗ **allocate_io_buffer** ()
- int **append_io_block_as_item** ()
- int **copy_item_to_io_block** ()
- int **extend_io_buffer** ()
- int **find_io_block** ()
- void **free_io_buffer** ()
- uintmax_t **get_count** ()
- uint16_t **get_count16** ()
- double **get_double** ()
- int32_t **get_int32** ()
- int **get_item_begin** ()
- int **get_item_end** ()
- long **get_long** ()
- int **get_long_string** ()
- double **get_real** ()
- intmax_t **get_scount** ()

- int16_t **get_scount16** ()
- int **get_short** ()
- int **get_string** ()
- uint32_t **get_uint32** ()
- int **get_var_string** ()
- void **get_vector_of_byte** ()
- void **get_vector_of_double** ()
- void **get_vector_of_float** ()
- void **get_vector_of_int** ()
- void **get_vector_of_int32** ()
- void **get_vector_of_long** ()
- void **get_vector_of_real** ()
- void **get_vector_of_short** ()
- void **get_vector_of_uint16** ()
- void **get_vector_of_uint32** ()
- int **list_io_blocks** ()
- int **list_sub_items** ()
- long **next_subitem_ident** ()
- long **next_subitem_length** ()
- int **next_subitem_type** ()
- void **put_count** ()
- void **put_count16** ()
- void **put_double** ()
- void **put_int32** ()
- int **put_item_begin** ()
- int **put_item_end** ()
- void **put_long** ()
- int **put_long_string** ()
- void **put_real** ()
- void **put_scount** ()
- void **put_scount16** ()
- void **put_short** ()
- int **put_string** ()
- void **put_uint32** ()
- int **put_var_string** ()
- void **put_vector_of_byte** ()
- void **put_vector_of_double** ()
- void **put_vector_of_float** ()
- void **put_vector_of_int** ()
- void **put_vector_of_int32** ()
- void **put_vector_of_long** ()
- void **put_vector_of_real** ()
- void **put_vector_of_short** ()
- void **put_vector_of_uint16** ()
- void **put_vector_of_uint32** ()
- int **read_io_block** ()
- int **remove_item** ()
- int **reset_io_block** ()
- int **rewind_item** ()
- int **search_sub_item** ()

- int **skip_io_block** ()
- int **skip_subitem** ()
- int **unget_item** ()
- int **unput_item** ()
- int **write_io_block** ()

8.8.1 Detailed Description

Author:

Konrad Bernloehr

Date:

1991 to 2000
CVS
Date
2006/02/27 11:15:21

Version:

CVS
Revision
1.8

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.

8.8.2 Define Documentation

8.8.2.1 #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)))
```

8.9 io_simtel.c File Reference

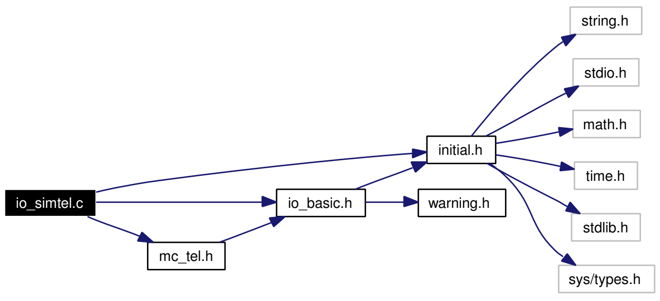
Write and read CORSIKA blocks and simulated Cherenkov photon bunches.

`#include "initial.h"`

`#include "io_basic.h"`

`#include "mc_tel.h"`

Include dependency graph for io_simtel.c:



Functions

- int begin_read_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int ∗array)

  Begin reading data for one array of telescopes/detectors.

- int begin_write_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih, int array)

  Begin writing data for one array of telescopes/detectors.

- int end_read_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih)

  End reading data for one array of telescopes/detectors.

- int end_write_tel_array (IO_BUFFER ∗iobuf, IO_ITEM_HEADER ∗ih)

  End writing data for one array of telescopes/detectors.

- int read_camera_layout (IO_BUFFER ∗iobuf, int max_pixels, int ∗itel, int ∗type, int ∗pixels, double ∗xp, double ∗yp)

  Read the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.

- int read_input_lines (IO_BUFFER ∗iobuf, struct linked_string ∗list)

  Read a block with several character strings (normally containing the text of the CORSIKA inputs file) into a linked list.

- int read_photo_electrons (IO_BUFFER ∗iobuf, int max_pixels, int max_pe, int ∗array, int ∗tel, int ∗npe, int ∗pixels, int ∗pe_counts, int ∗tstart, double ∗t)

  Read the photoelectrons registered in a Cherenkov telescope camera.

- int read_shower_longitudinal (IO_BUFFER ∗iobuf, int ∗event, int ∗type, double ∗data, int ndim, int ∗np, int ∗nthick, double ∗thickstep, int max_np)

  Read CORSIKA shower longitudinal distributions.

- int read_tel_block (IO_BUFFER ∗iobuf, int type, real ∗data, int maxlen)

  Read a CORSIKA header/trailer block of given type (see mc_tel.h).

- int read_tel_offset (IO_BUFFER ∗iobuf, int max_array, int ∗narray, double ∗toff, double ∗xoff, double ∗yoff)

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

- int read_tel_offset_w (IO_BUFFER ∗iobuf, int max_array, int ∗narray, double ∗toff, double ∗xoff, double ∗yoff, double ∗weight)

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

- int read_tel_photons (IO_BUFFER ∗iobuf, int max_bunches, int ∗array, int ∗tel, double ∗photons, struct bunch ∗bunches, int ∗nbunches)

  Read bunches of Cherenkov photons for one telescope/detector.

- int read_tel_pos (IO_BUFFER ∗iobuf, int max_tel, int ∗ntel, double ∗x, double ∗y, double ∗z, double ∗r)

  Read positions of telescopes/detectors within a system or array.

- int write_camera_layout (IO_BUFFER ∗iobuf, int itel, int type, int pixels, double ∗xp, double ∗yp)

  Write the layout (pixel positions) of a camera used for converting from photons to photo-electrons in a pixel.

- int write_input_lines (IO_BUFFER ∗iobuf, struct linked_string ∗list)

  Write a linked list of character strings (normally containing the text of the CORSIKA inputs file) as a dedicated block.

- int write_photo_electrons (IO_BUFFER ∗iobuf, int array, int tel, int npe, int pixels, int ∗pe_counts, int ∗tstart, double ∗t)

  Write the photo-electrons registered in a Cherenkov telescope camera.

- int write_shower_longitudinal (IO_BUFFER ∗iobuf, int event, int type, double ∗data, int ndim, int np, int nthick, double thickstep)

  Write CORSIKA shower longitudinal distributions.

- int write_tel_block (IO_BUFFER ∗iobuf, int type, int num, real ∗data, int len)

  Write a CORSIKA block as given type number (see mc_tel.h).

- int write_tel_compact_photons (IO_BUFFER ∗iobuf, int array, int tel, double photons, struct compact_bunch ∗cbunches, int nbunches, int ext_bunches, char ∗ext_fname)

  Write all the photon bunches for one telescope to an I/O buffer.

- int write_tel_offset (IO_BUFFER ∗iobuf, int narray, double toff, double ∗xoff, double ∗yoff)

  Write offsets of randomly scattered arrays with respect to shower core.

- int write_tel_offset_w (IO_BUFFER ∗iobuf, int narray, double toff, double ∗xoff, double ∗yoff, double ∗weight)

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

- int write_tel_photons (IO_BUFFER ∗iobuf, int array, int tel, double photons, struct bunch ∗bunches, int nbunches, int ext_bunches, char ∗ext_fname)

  Write all the photon bunches for one telescope to an I/O buffer.

- int write_tel_pos (IO_BUFFER ∗iobuf, int ntel, double ∗x, double ∗y, double ∗z, double ∗r)

  Write positions of telescopes/detectors within a system or array.

8.9.1 Detailed Description

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

Author:
Konrad Bernloehr

Date:
1997, 2000
CVS
Date
2005/04/06 12:27:19

Version:
CVS
Revision
1.7

8.9.2 Function Documentation

8.9.2.1 int begin_read_tel_array (IO_BUFFER $*$ *iobuf*, IO_ITEM_HEADER $*$ *ih*, int $*$ *array*)

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

Parameters:

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)

8.9.2.2 int begin_write_tel_array (IO_BUFFER $*$ *iobuf*, IO_ITEM_HEADER $*$ *ih*, int *array*)

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

Parameters:

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)

8.9.2.3 int end_read_tel_array (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *ih*)

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)

8.9.2.4 int end_write_tel_array (IO_BUFFER ∗ *iobuf*, IO_ITEM_HEADER ∗ *ih*)

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)

8.9.2.5 int read_camera_layout (IO_BUFFER ∗ *iobuf*, int *max_pixels*, int ∗ *itel*, int ∗ *type*, int ∗ *pixels*, double ∗ *xp*, double ∗ *yp*)

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)

8.9.2.6 int read_input_lines (IO_BUFFER ∗ *iobuf*, struct linked_string ∗ *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)

8.9.2.7 int read_photo_electrons (IO_BUFFER ∗ *iobuf*, int *max_pixels*, int *max_pe*, int ∗ *array*, int ∗ *tel*, int ∗ *npe*, int ∗ *pixels*, int ∗ *pe_counts*, int ∗ *tstart*, double ∗ *t*)

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.

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.

Returns:

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

8.9.2.8 int read_shower_longitudinal (IO_BUFFER ∗ *iobuf*, int ∗ *event*, int ∗ *type*, double ∗ *data*, int *ndim*, int ∗ *np*, int ∗ *nthick*, double ∗ *thickstep*, int *max_np*)

See 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)

8.9.2.9 int read_tel_block (IO_BUFFER ∗ *iobuf*, int *type*, real ∗ *data*, int *maxlen*)

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)

8.9.2.10 int read_tel_offset (IO_BUFFER ∗ *iobuf*, int *max_array*, int ∗ *narray*, double ∗ *toff*, double ∗ *xoff*, double ∗ *yoff*)

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)

8.9.2.11 int read_tel_offset_w (IO_BUFFER ∗ *iobuf*, int *max_array*, int ∗ *narray*, double ∗ *toff*, double ∗ *xoff*, double ∗ *yoff*, double ∗ *weight*)

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)

8.9.2.12 int read_tel_photons (IO_BUFFER ∗ *iobuf*, int *max_bunches*, int ∗ *array*, int ∗ *tel*, double ∗ *photons*, struct bunch ∗ *bunches*, int ∗ *nbunches*)

The data format may be either the more or less compact one.

Parameters:

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)

8.9.2.13 int read_tel_pos (IO_BUFFER ∗ *iobuf*, int *max_tel*, int ∗ *ntel*, double ∗ *x*, double ∗ *y*, double ∗ *z*, double ∗ *r*)

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)

8.9.2.14 int write_camera_layout (IO_BUFFER ∗ *iobuf*, int *itel*, int *type*, int *pixels*, double ∗ *xp*, double ∗ *yp*)

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)

8.9.2.15 int write_input_lines (IO_BUFFER ∗ *iobuf*, struct linked_string ∗ *list*)

Parameters:

iobuf	I/O buffer descriptor
list	starting point of linked list

Returns:

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

8.9.2.16 int write_photo_electrons (IO_BUFFER ∗ *iobuf*, int *array*, int *tel*, int *npe*, int *pixels*, int ∗ *pe_counts*, int ∗ *tstart*, double ∗ *t*)

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

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.

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

8.9.2.17 int write_shower_longitudinal (IO_BUFFER ∗ *iobuf*, int *event*, int *type*, double ∗ *data*, int *ndim*, int *np*, int *nthick*, double *thickstep*)

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)

8.9.2.18 int write_tel_block (IO_BUFFER ∗ *iobuf*, int *type*, int *num*, real ∗ *data*, int *len*)

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)

8.9.2.19 int write_tel_compact_photons (IO_BUFFER ∗ *iobuf*, int *array*, int *tel*, double *photons*, struct compact_bunch ∗ *cbunches*, int *nbunches*, int *ext_bunches*, char ∗ *ext_fname*)

Usually, calls to this function for each telescope in an array should be enclosed within calls to begin_-write_tel_array() and end_write_tel_array(). This routine writes the more compact format (16 bytes per bunch). The more compact format should usually be used to save memory and disk space.

Parameters:

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)

8.9.2.20 int write_tel_offset (IO_BUFFER ∗ *iobuf*, int *narray*, double *toff*, double ∗ *xoff*, double ∗ *yoff*)

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)

8.9.2.21 int write_tel_offset_w (IO_BUFFER ∗ *iobuf*, int *narray*, double *toff*, double ∗ *xoff*, double ∗ *yoff*, double ∗ *weight*)

With respect to the backwards-compatible non-weights version write_tel_offset(), this version adds a weight to each offset position which should be normalized in such a way that with uniform sampling it should be the area over which showers are thrown divided by the number of array in each shower. With importance sampling the same relation should hold on average. So in either case, the average sum of weights for the different offsets in one shower equals just the area over which cores are randomized. This leaves the possibility to change the number of offsets from shower to shower.

Parameters:

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)

8.9.2.22 int write_tel_photons (IO_BUFFER ∗ *iobuf*, int *array*, int *tel*, double *photons*, struct bunch ∗ *bunches*, int *nbunches*, int *ext_bunches*, char ∗ *ext_fname*)

Usually, calls to this function for each telescope in an array should be enclosed within calls to begin_-write_tel_array() and end_write_tel_array(). This routine writes the less compact format (32 bytes per bunch).

Parameters:

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)

8.9.2.23 int write_tel_pos (IO_BUFFER ∗ *iobuf*, int *ntel*, double ∗ *x*, double ∗ *y*, double ∗ *z*, double ∗ *r*)

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)

8.10 listio.c File Reference

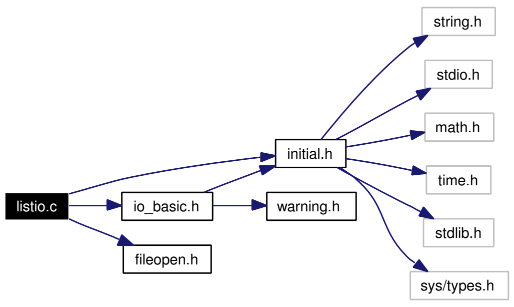
Main function for listing data consisting of eventio blocks.

`#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.

8.10.1 Detailed Description

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.

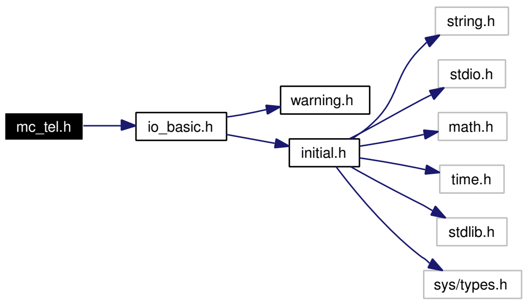
```
Syntax: listio [-s[n]] [-p] [filename]
List structure of eventio data files.
   -s : also list contained (sub-) items
   -sn: list sub-items up to depth n (n=0,1,...)
   -p : show positions of items in the file
If no file name given, standard input is used.
```

8.11 mc_tel.h File Reference

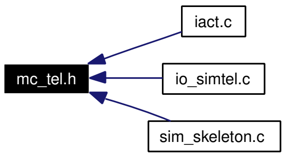
Definitions and structures for CORSIKA Cherenkov light interface.

`#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 linked_string

  The linked_string is mainly used to keep CORSIKA input.

- struct photo_electron

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

Defines

- #define **_MC_TEL_LOADED** 1
- #define **IO_TYPE_MC_BASE** 1200
- #define **IO_TYPE_MC_EVTE** (IO_TYPE_MC_BASE+9)
- #define **IO_TYPE_MC_EVTH** (IO_TYPE_MC_BASE+2)
- #define **IO_TYPE_MC_INPUTCFG** (IO_TYPE_MC_BASE+12)
- #define **IO_TYPE_MC_LAYOUT** (IO_TYPE_MC_BASE+6)
- #define **IO_TYPE_MC_LONGI** (IO_TYPE_MC_BASE+11)
- #define **IO_TYPE_MC_PE** (IO_TYPE_MC_BASE+8)
- #define **IO_TYPE_MC_PHOTONS** (IO_TYPE_MC_BASE+5)
- #define **IO_TYPE_MC_RUNE** (IO_TYPE_MC_BASE+10)
- #define **IO_TYPE_MC_RUNH** (IO_TYPE_MC_BASE+0)
- #define **IO_TYPE_MC_TELARRAY** (IO_TYPE_MC_BASE+4)
- #define **IO_TYPE_MC_TELOFF** (IO_TYPE_MC_BASE+3)
- #define **IO_TYPE_MC_TELPOS** (IO_TYPE_MC_BASE+1)
- #define **IO_TYPE_MC_TRIGTIME** (IO_TYPE_MC_BASE+7)

Typedefs

- typedef short **INT16**
- typedef int **INT32**
- typedef float **real**
- typedef unsigned short **UINT16**
- typedef unsigned int **UINT32**

Functions

- int **begin_read_tel_array** ()
- int **begin_write_tel_array** ()
- int **end_read_tel_array** ()
- int **end_write_tel_array** ()
- int **read_camera_layout** ()
- int **read_input_lines** ()
- int **read_photo_electrons** ()
- int **read_shower_longitudinal** ()
- int **read_tel_block** ()
- int **read_tel_offset** ()
- int **read_tel_offset_w** ()
- int **read_tel_photons** ()
- int **read_tel_pos** ()
- int **write_camera_layout** ()
- int **write_input_lines** ()
- int **write_photo_electrons** ()
- int write_shower_longitudinal (IO_BUFFER ∗iobuf, int event, int type, double ∗data, int ndim, int np, int nthick, double thickstep)

  Write CORSIKA shower longitudinal distributions.

- int **write_tel_block** ()
- int **write_tel_compact_photons** ()
- int **write_tel_offset** ()
- int **write_tel_offset_w** ()
- int **write_tel_photons** ()
- int **write_tel_pos** ()

8.11.1 Detailed Description

This file contains definitions of data structures and of function prototypes as needed for the Cherenkov light extraction interfaced to the modified CORSIKA code.

Author:

Konrad Bernloehr

Date:

1997

CVS

Date

2003/11/12 19:22:55

Version:

CVS

Revision

1.5

8.11.2 Function Documentation

8.11.2.1 int write_shower_longitudinal (IO_BUFFER ∗ *iobuf*, int *event*, int *type*, double ∗ *data*, int *ndim*, int *np*, int *nthick*, double *thickstep*)

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:

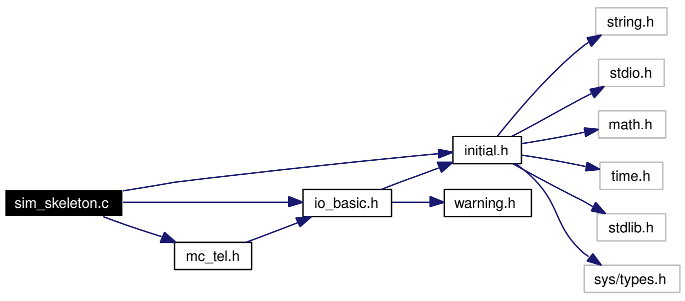
0 (o.k.), -1, -2, -3 (error, as usual in eventio)

8.12 sim_skeleton.c File Reference

A (non-functional) skeleton program for reading CORSIKA IACT data.

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

Include dependency graph for sim_skeleton.c:



Data Structures

- struct camera_electronics

  Parameters of the electronics of a telescope.

- struct mc_run

  Basic parameters of the CORSIKA run.

- struct pm_camera

  Parameters of a telescope camera (pixels, .

- struct simulated_shower_parameters

  Basic parameters of a simulated shower.

- struct telescope_array

  Description of telescope position, array offets and shower parameters.

- struct telescope_optics

  Parameters describing the telescope optics.

Defines

- #define MAX_ARRAY 100

  The largest no.

- #define **MAX_BUNCHES** 50000
- #define **MAX_PHOTOELECTRONS** 100000
- #define MAX_PIXELS 1024

  The largest no.

- #define MAX_TEL 16

  The largest no.

- #define Nair(hkm) (1.+0.0002814∗exp(-0.0947982∗(hkm)-0.00134614∗(hkm)∗(hkm)))

  Refraction index of air as a function of height in km (0km<=h<=8km).

Functions

- double **atmospheric_transmission** (int iwl, double zem, double airmass)
- void atmset_ (int ∗iatmo, double ∗obslev)

  Set number of atmospheric model profile to be used.

- double **heigh_** (double x)
- double line_point_distance (double x1, double y1, double z1, double cx, double cy, double cz, double x, double y, double z)

  Distance between a straight line and a point in space.

- int main (int argc, char ∗∗argv)

  Main program of Cherenkov telescope simulation.

- double **RandFlat** (void)
- double **rhof_** (double h)
- double **thick_** (double h)

Variables

- static double **airlightspeed** = 29.9792458/1.0002256
- linked_string **corsika_inputs**

8.12.1 Detailed Description

Copyright by Konrad Bernloehr (1997, 1999). All rights reserved. This file may be modified but all modified version must be declared as modified and by whom they were modified.

This file contains a (non-functional) skeleton of the telescope simulation. It serves only as an illustration of the essential usage of CORSIKA related eventio functions to read CORSIKA data in eventio format and how some of the required values are extracted. Comment lines with '...' usually indicate that you should fill in relevant code yourself.

This file comes with no warranties.

8.12.2 Define Documentation

8.12.2.1 #define MAX_ARRAY 100

of arrays to be handled

8.12.2.2 #define MAX_PIXELS 1024

of pixels per camers

8.12.2.3 #define MAX_TEL 16

of telescopes/array.

8.12.3 Function Documentation

8.12.3.1 void atmset_ (int $*$ *iatmo*, double $*$ *obslev*)

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

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

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

Parameters:

iatmo (pointer to) atmospheric profile number; negative for CORSIKA built-in profiles.

obslev (pointer to) altitude of observation level [cm]

Returns:

(none)

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

Parameters:

x1,y1,z1 reference point on the line

cx,cy,cz direction cosines of the line

x,y,z point in space

Returns:

distance

8.13 straux.c File Reference

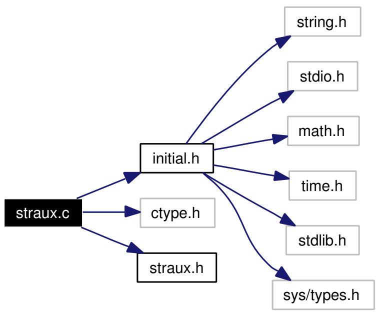
Check for abbreviations of strings and get words from strings.

`#include "initial.h"`

`#include <ctype.h>`

`#include "straux.h"`

Include dependency graph for straux.c:



Defines

- #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.

8.13.1 Detailed Description

Author:

Konrad Bernloehr

Date

2003/09/12 21:09:44

Revision

1.2

8.13.2 Function Documentation

8.13.2.1 int abbrev (CONST char ∗ *s*, CONST char ∗ *t*)

s may be an abbreviation of t. Upper/lower case in s is ignored. s has to be at least as long as the leading upper case, digit, and '_' part of t.

Parameters:

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.

8.13.2.2 int getword (CONST char ∗ *s*, int ∗ *spos*, char ∗ *word*, int *maxlen*, char *blank*, char *endchar*)

The word must have a length less than or equal to maxlen.

Parameters:

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

8.13.2.3 int stricmp (CONST char ∗ *a*, CONST char ∗ *b*)

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

8.14 testio.c File Reference

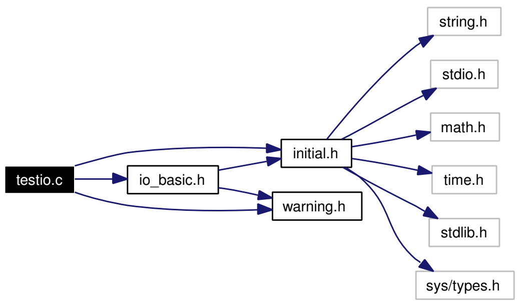
Test program for eventio data format.

`#include "initial.h"`

`#include "warning.h"`

`#include "io_basic.h"`

Include dependency graph for testio.c:



Data Structures

- struct **test_struct**

Typedefs

- typedef test_struct **TEST_DATA**

Functions

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

    Compare elements of test data structures.

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

    Main function for I/O test program.

- int read_test1 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

    Read test data with single-element functions.

- int **read_test1** ()
- int read_test2 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

    Read test data with vector functions as far as possible.

- int **read_test2** ()
- int write_test1 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

    Write test data with single-element functions.

- int **write_test1** ()

- int write_test2 (TEST_DATA ∗data, IO_BUFFER ∗iobuf)

  Write test data with vector functions as far as possible.

- int **write_test2** ()

Variables

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

8.14.1 Detailed Description

Author:

Konrad Bernloehr

Date:

1994, 1997, 2000

CVS

Date

2006/02/27 11:15:21

Version:

CVS

Revision

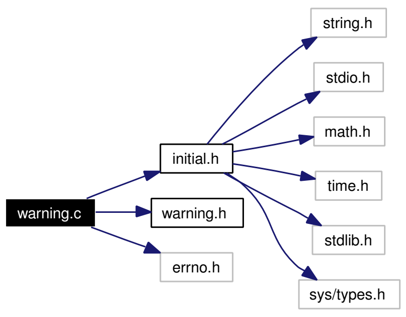
1.11

8.15 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.

Defines

- #define **__WARNING_MODULE** 1
- #define **get_warn_specific**() (&warn_defaults)

Functions

- void flush_output ()

  Flush buffered output.

- void set_aux_warning_function (char ∗(∗auxfunc)())

  Set an auxilliary function for warnings.

- void **set_default_aux_warning_function** (char ∗(∗auxfunc)())
- void **set_default_logging_function** (void(∗user_function)())
- void **set_default_output_function** (void(∗user_function)())
- int **set_default_warning** (int level, int mode)
- int set_log_file (const char ∗fname)

  Set a new log file name and save it in local storage.

- void set_logging_function (void(∗user_function)())

  Set user-defined function for logging warnings and errors.

- void set_output_function (void(∗user_function)())

  Set a user-defined function as the function to be used for normal text output.

- int set_warning (int level, int mode)

  Set a specific warning level and mode.

- void warn_f_output_text (const char ∗text)

  Print a text string (without appending a newline etc.

- void warn_f_warning (const char ∗msgtext, const char ∗msgorigin, int msglevel, int msgno)

  Issue a warning to screen or other configured target.

- void warning_status (int ∗plevel, int ∗pmode)

  Inquire status of warning settings.

Variables

- static struct warn_specific_data **warn_defaults**

8.15.1 Detailed Description

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().
------------------------------------------------------------
```

8.15.2 Function Documentation

8.15.2.1 void flush_output ()

Output is flushed, no matter if it is standard output or a special output function;

Returns:
(none)

8.15.2.2 void set_aux_warning_function (char ∗(∗)() *auxfunc*)

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)

8.15.2.3 int set_log_file (const char ∗ *fname*)

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)

8.15.2.4 void set_logging_function (void(∗)() *user_function*)

Set a user-defined function as the function to be used for logging warnings and errors. To enable usage of this function, bit 2 of the warning mode must be set and other bits reset, if logging to screen and/or disk file is no longer wanted.

Parameter userfunc: Pointer to a function taking two strings (the message text and the origin text, which may be NULL) and two integers (message level and message number).

Returns:
(none)

8.15.2.5 void set_output_function (void($*$)() *user_function*)

Such a function may be used to send output back to a remote control process via network.

Parameter userfunc: Pointer to a function taking a string (the text to be displayed) as argument.

Returns:
(none)

8.15.2.6 int set_warning (int *level*, int *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.

8.15.2.7 void warn_f_output_text (const char $*$ *text*)

) 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)

8.15.2.8 void warn_f_warning (const char $*$ *msgtext*, const char $*$ *msgorigin*, int *msglevel*, int *msgno*)

Issue a warning to screen and/or file if the warning has a sufficiently large message 'level' (high enough severity). This function should best be called through the macros 'Information', 'Warning', and 'Error'. The name of this function has been changed from 'warning' to '_warning' to avoid trouble if you call 'warning' instead of 'Warning'. Now such a typo causes an error in the link step.

Parameters:
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)

8.15.2.9 void warning_status (int ∗ *plevel*, int ∗ *pmode*)

Parameters:

plevel Pointer to variable for storing current level.

pmode Pointer to store the current warning mode.

Returns:

(none)

8.15.3 Variable Documentation

8.15.3.1 struct warn_specific_data warn_defaults `[static]`

Initial value:

```
{
   0,
   1+8,
   "",
   "warning.log",
   "",
   0,
   NULL,
   NULL,
   NULL,
   NULL,
   0
}
```

Index