AIRES

A system for air shower simulations

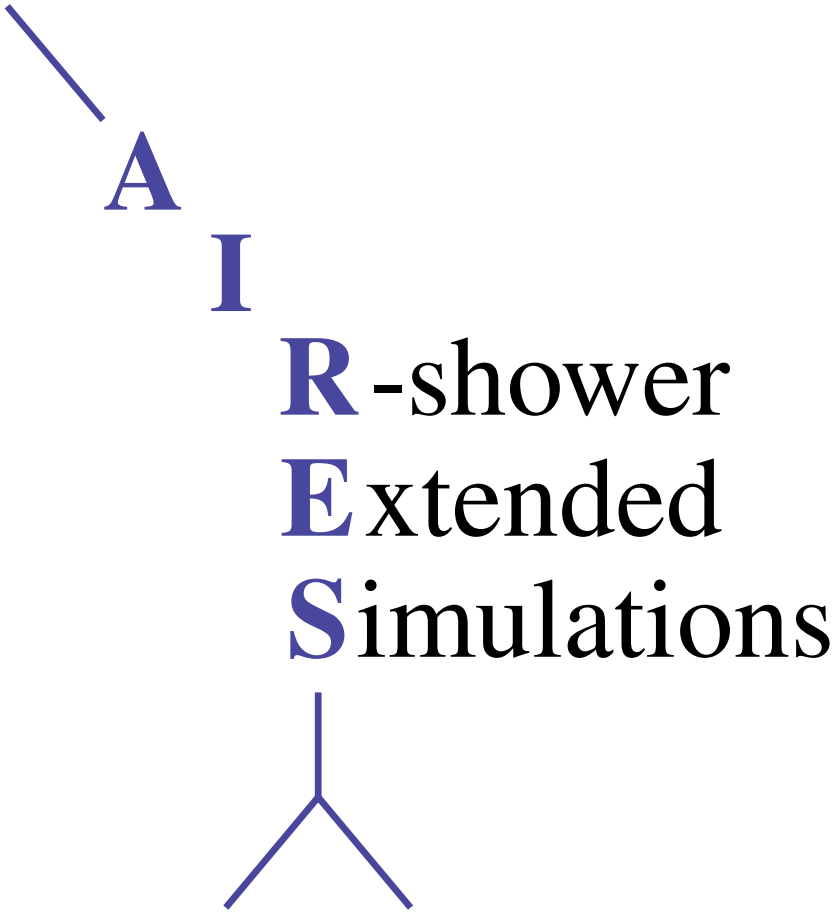
User's guide and reference manual

Version 19.04.08

S. J. Sciutto

Departamento de Física and IFLP (CONICET)
Universidad Nacional de La Plata
C. C. 67 - 1900 La Plata
Argentina
sciutto@fisica.unlp.edu.ar

October 22, 2021


AIRES
AIR-shower
Extended
Simulations

AIRES user's guide and reference manual
Version 19.04.08 (2021)
S. J. Sciutto, La Plata, Argentina.

Summary

The name **AIRES** (**AIR**-shower **E**xtended **S**imulations) identifies a set of programs and subroutines to simulate particle showers produced after the incidence of high energy cosmic rays on the Earth's atmosphere, and to manage and analyze all the related output data.

AIRES provides full space-time particle propagation in a realistic environment, where the characteristics of the atmosphere, the geomagnetic field and the Earth's curvature are taken into account adequately. A statistical sampling procedure (the so-called *thinning*) is used when the number of particles in the showers is exceedingly large. The thinning algorithms used in AIRES are unbiased, that is, the statistical sampling never alters the average values of output observables.

The particles taken into account by AIRES in the simulations are: Gammas; electrons; positrons; muons; taus; pions; kaons; eta, rho, D, D_s, and B mesons; lambda, sigma, xi, and omega baryons; nucleons; antinucleons; and nuclei up to $Z = 36$. Electron and muon neutrinos are generated in certain processes (decays) and accounted for their energy, but not propagated. The primary particle can be any one of the already mentioned particles, with energy ranging from less than 1 GeV up to more than 1 ZeV (10^{21} eV). It is also possible to simulate showers initiated by "special" primary particles via a call to a user-written module capable of processing the "first interaction" of the primary and returning a list of standard particles suitable for being processed by AIRES.

Among all the physical processes that may undergo the shower particles, the most important from the probabilistic point of view are taken into account in the simulations. Such processes are: (i) *Electrodynamical processes:* Pair production and electron-positron annihilation, bremsstrahlung (electrons, positrons and muons), muonic pair production, knock-on electrons (δ rays), Compton and photoelectric effects, Landau-Pomeranchuk-Migdal (LPM) effect and dielectric suppression. (ii) *Unstable particle decays,* pions and muons, for instance. (iii) *Hadronic processes:* Inelastic collisions hadron-nucleus and photon-nucleus, sometimes simulated using an external package which implements a given hadronic interaction model, like the well-known EPOS, QGSJET or SIBYLL models. Photonuclear reactions. Nuclear fragmentation, elastic and inelastic. (iv) *Propagation of charged particles:* Losses of energy in the medium (ionization), multiple Coulomb scattering and geomagnetic deflections.

The AIRES simulation system provides a comfortable environment where to perform reliable simulations: The *Input Directive Language (IDL)* is a set of simple directives which allow for an efficient control of the input parameters of the simulation. The *AIRES Runner System* is a powerful tool to manage long simulation tasks in UNIX environments, allowing the user to coordinate several

tasks running concurrently, controlling the evolution of a given job while running, etc. The *AIRES summary program* processes the internal dump files generated by the main simulation program, and allows to obtain data related with physical observables either after or during the simulations. Finally, the *AIRES object library* provides a series of auxiliary routines to process the data generated by the simulation program, in particular the data contained in the *compressed output files,* the detailed particle data files containing per-particle information for particles reaching the ground, crossing different observing levels during their evolution, etc.

The present version of AIRES (19.04.08) represents a new release of the Air Shower Simulation System where many new technical features have been added to it. In particular, the present version includes the possibility of installing *extensions,* additional modules that are incorporated to the main simulation system that enlarge its capabilities. This is the case of the *ZHAireS* extension, that allows to simulate the radio waves emitted during the shower development.

The present version of AIRES (19.04.08) represents a new release of the Air Shower Simulation System where many new features and algorithm improvements have been added to it. **The most important additions for this new version are summarized in next section *Release Notes.***

It is worthwhile mentioning that some of the modules that that make up the AIRES simulation engine have been developed with the help of various colleagues: The routines that simulate the geomagnetic deflections, LPM effect and muon bremsstrhlung and pair production were developed with the help of A. Cillis (La Plata, 1998-2001). The atmospheric model GAMMA was designed and developed in collaboration with J. C. Moreno (La Plata, 2008-2012). The HQIP preprocessor was designed and developed in collaboration with J. I. Illana and M. Masip (Universidad de Granada, Spain, 2010-2014). The updated cross sections for photonuclear interactions have been investigated in collaboration with C. A. Garcıa Canal (La Plata), G. Pancheri (INFN Frascati, Italy), and F. Cornet and A. Grau (Universidad de Granada, Spain, 2008-2015). Additionally, many of the developments presented for the current release were performed taking into account users' suggestions and remarks. The author is indebted to everybody that have contacted him (a very long list of persons indeed), either to report a bug or to make a comment on the program.

La Plata, October 2021.

Release notes

We include here a brief summary of the new developments that are included in the current version of AIRES (19.04.08), as well as a description of the differences with the previous release of the system (19.04.06).

The number in brackets placed after directive names or library routines indicates the page where the corresponding directive/routine is described. Example: **TaskName** (130).

Differences between AIRES 19.04.08 and AIRES 19.04.06

Bugs

Several problems with AIRES 19.04.06simulation program were detected or reported by several users. All the errors in the program's code –mostly minor bugs– were fixed and are no longer present in the current version of AIRES. Some of those bugs are:

- Correction of bugs related with the interface of AIRES with the hadronic collision packages EPOS and SIBYLL (bug found thanks to Jaime Alvarez-Muñiz, Roberta Colalillo, Eva Santos, and Matías Tueros).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cosmic rays with energies larger than 100 TeV must be studied –at present– using experimental devices located on the surface of the Earth. This implies that such kind of cosmic rays cannot be detected directly; it is necessary instead to measure the products of the atmospheric cascades of particles initiated by the incident astroparticle. An atmospheric particle shower begins when the primary cosmic particle interacts with the Earth's atmosphere. This is, in general, an inelastic nuclear collision that generates a number of secondary particles. Those particles continue interacting and generating more secondary particles which in turn interact again similarly as their predecessors. This multiplication process continues until a maximum is reached. Then the shower attenuates as far as more and more particles fall below the threshold for further particle production.

A detailed knowledge of the physics involved is thus necessary to interpret adequately the measured observables and be able to infer the properties of the primary particles. This is a complex problem involving many aspects: Interactions of high energy particles, properties of the atmosphere and the geomagnetic field, etc. Computer simulation is one of the most convenient tools to quantitatively analyze such particle showers.

In the case of air showers initiated by ultra-high energy astroparticles ($E \geq 10^{19}$ eV), the primary particles have energies that are several orders of magnitude larger than the maximum energies attainable in experimental colliders. This means that the models used to rule the behavior of such energetic particles must necessarily make extrapolations from the data available at much lower energies, and there is still no definitive agreement about what is the most convenient model to accept among the several available ones.

The **AIRES system**[1] is a set of programs to simulate such air showers. One of the basic objectives considered during the development of the software is that of designing the program modularly, in order to make it easier to switch among the different models that are available, without having to get attached to a particular one.

Several simulation programs that were developed in the past were studied in detail in order to gain experience and improve the new design. Among such programs, the MOCCA code created by A. M.

[1] **AIRES** is an acronym for **AIR**-shower **E**xtended **S**imulations.

Hillas [1] has been extensively used as the primary reference when developing the first version of AIRES [2] released in May 1997. Needless to say, the present version of AIRES (19.04.08), released more that 20 years after the first one (1.2.0) does include modifications, sometimes massive, of the original algorithms, and in consequence both programs are no longer equivalent.

Another characteristic of ultra-high energy simulations that was taken into account when developing AIRES is the large number of particles involved. For example, a 10^{20} eV shower contains about 10^{11} secondary particles. From the computational point of view, this fact has two main consequences that were specially considered at the moment of designing AIRES: (i) With present day computers, it is virtually impossible to follow all the generated particles, and therefore a suitable sampling technique must be used to reduce the number of particles actually simulated. The so-called thinning algorithm introduced by Hillas [4] or the sampling algorithm of Kobal, Filipčič and Zavrtanik [5] represent examples of such sampling methods. (ii) The simulation algorithm is CPU intensive, and therefore it is necessary to develop a series of special procedures that will provide an adequate environment to process computationally long tasks.

There are many quantities that define the initial or environmental conditions for an air shower, for example, the identity of the primary particle and its energy, the position of the ground surface, the minimum energy a particle must have to be taken into account in the simulation, the intensity and orientation of the geomagnetic field, etc. Additionally, it is possible to define many observables that are useful to characterize the particle shower, namely, longitudinal and lateral distribution of particles, energy distributions, position of the shower maximum and so on.

A comfortable environment is provided by AIRES to manage all the input and output data: The *Input Directive Language (IDL)* is a set of human-readable input directives that allow the user to efficiently steer the simulations. The *AIRES summary program* and the *AIRES object library* represent a set of tools to manage the output data after the simulations are finished, and even during them, allowing to control their evolution. Data associated with particles reaching ground or crossing predetermined observing levels can be recorded into *compressed output files.* A special data compression procedure is used to reduce as much as possible the size of the files, which tends to be very large in certain circumstances. The compressed files can be processed with the help of some auxiliary routines that are included in the AIRES library. The machine and operating system used to generate such files may be different that the ones used to read them.

The particles taken into account by AIRES in the simulations are: Gammas; electrons; positrons; taus; muons; pions; kaons; eta, rho, D, D_s, and B mesons; lambda, sigma; xi, and omega baryons; nucleons; antinucleons; and nuclei up to $Z = 36$. Electron and muon neutrinos are generated in certain processes (decays) and accounted for their energy, but not propagated. The primary particle can be any one of the already mentioned particles, with energy ranging from less than 1 GeV up to more than 1 ZeV (10^{21} eV). It is also possible to simulate showers initiated by "special" primary particles via a call to a user-written module capable of processing the "first interaction" of the primary and returning a list of standard particles suitable for being processed by AIRES. A detailed description on how to define and use special primaries is placed in section 3.5.

Among all the physical processes that may undergo the shower particles, the most important

from the probabilistic point of view are taken into account in the simulations. Such processes are: (i) *Electrodynamical processes:* Pair production and electron-positron annihilation, bremsstrahlung (electrons, positrons and muons), muonic pair production, knock-on electrons (δ rays), Compton and photoelectric effects, Landau-Pomeranchuk-Migdal (LPM) effect and dielectric suppression. (ii) *Unstable particle decays,* pions and muons, for instance. (iii) *Hadronic processes:* Inelastic collisions hadron-nucleus and photon-nucleus, generally simulated using an external package which implements a given hadronic interaction model like the well-known EPOS [6], QGSJET [7], or SIBYLL [10] models; or by a built-in algorithm called *extended Hillas splitting algorithm* (EHSA). Photonuclear reactions. Nuclear fragmentation, elastic and inelastic. (iv) *Propagation of charged particles:* Losses of energy in the medium (ionization), multiple Coulomb scattering and geomagnetic deflections.

All the general characteristics of AIRES and the physics involved in air shower simulations are summarized in table 1.1; they are described in more detail in chapter 2.

AIRES is completely written in standard FORTRAN (using a few extensions that are, to the best of our knowledge, accepted by all FORTRAN compilers). The complete AIRES 19.04.08 source code, which includes the EPOS LHC [6], EPOS 1.99 [6], QGSJET-II-03 [8], QGSJET-II-04 [7], SIBYLL 2.3c [10], SIBYLL 2.3 [11], and SIBYLL 2.1 [12] hadronic collisions packages, the IGRF [15] routines to evaluate geomagnetic data and Netlib/minpack/lmder nonlinear least squares fitting package [16], consists of more than 2000 routines, adding up to more than 300,000 source lines extensively commented.

In the present version, the AIRES simulation system consists of the following:

- The main air shower simulation programs:
  - **AiresEPLHC** and **AiresEP199**;
  - **AiresQIIr04** and **AiresQIIr03**;
  - **AiresS23**, **AiresS23c**, and **AiresS21**;

  containing the interfaces with the hadronic collision packages EPOS LHC [6], EPOS 1.99 [6], QGSJET-II-04 [7], QGSJET-II-03 [8], SIBYLL 2.3 [11], SIBYLL 2.3c [10], and SIBYLL 2.1 [12], respectively. The default simulation program, **Aires**, is equivalent to **AiresS23**.

- The summary program (**AiresSry**) designed to process a part of the data generated by the simulation programs, allowing the user to analyze the results of the simulation after completing it, or even while it is being run.

- The IDF to ADF file format converting program **AiresIDF2ADF**.

- A library of utilities to help the user to process the compressed output data files generated by the simulation program, write external modules to process special primaries, etc. In LINUX environments this library is implemented as an object library called **libAires.a**, or a shared object library called **libAires.so**.

Propagated particles	Gammas. Leptons: $e^{\pm}$, $\mu^{\pm}$, $\tau^{\pm}$. Mesons: π^0, $\pi^{\pm}$; η, $K^0_{L,S}$, $K^{\pm}$, ρ^0, $\rho^{\pm}$, D^0, $D^{\pm}$, $D^{\pm}_s$, B^0, $B^{\pm}$. Baryons: p, $\bar{p}$, n, $\bar{n}$, Λ, Λ_c, Σ^0, $\Sigma^{\pm}$, Ξ^0, Ξ^-, Ω^-. Nuclei up to $Z = 36$. Neutrinos are generated (in decays) and accounted for their number and energy, but not propagated.
Primary particles	All propagated particles can be injected as primary particles. Multiple and/or "exotic" primaries can be injected using the *special primary* feature.
Primary energy range	From 500 MeV to 3 ZeV (3×10^{21} eV).
Geometry and environment	Incidence angles from vertical to horizontal showers. The Earth's curvature is taken into account for all inclinations. Realistic atmosphere. Geomagnetic deflections: The geomagnetic field can be calculated using the IGRF model [15].
Propagation (general)	Medium energy losses (ionization). Scattering of all charged particles including corrections for finite nuclear size. Geomagnetic deflections.
Propagation: *Electrons and gammas*	Compton and photoelectric effects, e^+e^- pair production. Bremsstrahlung, emission of knock-on electrons, and e^+ annihilation. LPM effect, and dielectric suppression. Photonuclear reactions.
Propagation: *Muons*	Bremsstrahlung and muonic pair production. Emission of knock-on electrons. Decay.
Propagation: *Hadrons and nuclei*	Hadronic collisions using the EHSA (low energy) and EPOS, QGSJET or SIBYLL (high energy). Nuleus-nucleus collisions via EPOS, QGSJET or SIBYLL, or using a built-in nuclear fragmentation algorithm. Hadronic cross sections are evaluated from fits to experimental data (low energy), or to EPOS, QGSJET or SIBYLL predictions (high energy). Emission of knock-on electrons. Decay of unstable hadrons.
Statistical sampling	Particles are sampled by means of the Hillas thinning algorithm [4], extended to allow control of maximum weights.
Main observables	Longitudinal development of all particles recorded in up to 510 observing levels. Energy deposited in the atmosphere. Lateral, energy and time distributions at ground level. Detailed list of particles reaching ground, and/or crossing predetermined observing levels.

Table 1.1. *Main characteristics of the AIRES air shower simulation system.*

- The AIRES runner system: A set of shell scripts to ease working with AIRES in UNIX environments.

1.1 Structure of the main simulation programs

An air shower starts when a cosmic particle reaches the Earth's atmosphere and interacts with it. In most cases the first interaction is an inelastic collision of the (high energy) primary particle with an air nucleus. The product of this collision is a set of secondary particles carrying a fraction of the primary's energy. These secondaries begin to move through the atmosphere and will eventually interact similarly as the primary did, generating new sets of secondaries. This multiplication process continues until a maximum is reached. After that moment the shower begins to attenuate because an increasing number of secondaries are produced with energies too low for further particle generation.

This phenomenon is simulated in AIRES in the following way:

1. Several data arrays or *stacks* are defined. Every record within any stacks is a particle entry, and represents a physical particle. The data contained in every record are related to the characteristics of the corresponding particle: Identity, position, energy, etc.

2. The particles can move inside a volume within the atmosphere where the shower takes place. This volume is limited by the *ground,* and *injection* surfaces, and by vertical planes which limit the region of interest.

3. Before starting the simulations all the stacks are empty. The first action is to add the first stack entry, which corresponds to the primary particle. The primary is initially located at the injection surface, and its downwards direction of motion defines the sower axis.

4. The stack entries are repeatedly processed sequentially. Every particle entry is updated analyzing first all the possible interactions it can have, and evaluating the corresponding probabilities for each possibility, taking into account the physics involved.

5. Using a stochastic method, the mentioned probabilities are used to select one of the possible interactions. This selection defines what is going to happen with the corresponding particle at that moment.

6. The interaction is processed: First the particle is moved a certain distance (which comes out from the mentioned stochastic method), then the products of the interaction are generated. New stack entries are appended to the existing lists for every one of the secondary particles that are created. Depending on the particular interaction that is being processed, the original particle may survive (the corresponding entry remains in the stack for further processing) or not (the entry is deleted).

7. When a charged particle is moved, its energy is modified to take into account the energy losses in the medium (ionization).

8. Particle entries can also be removed when one of the following events happens: (a) The energy of the particle is lower than a certain threshold energy called *cut energy.* The cut energies may be different for different particle kinds. (b) The particle reaches ground level. (c) A particle going upwards reaches the injection surface. (d) A particle with quasi-horizontal motion exits the region of interest.

9. After having scanned all the stacks, it is checked whether or not there are remaining particle entries pending further processing. If the answer is positive, then all the stacks are re-scanned once more; otherwise the simulation of the shower is complete.

The group of algorithms related with interaction selection and processing, as well as calculation of energy losses is the group of *physical algorithms.*

The most important air shower observables are those related with statistical distributions of particle properties. To evaluate such quantities the simulation engine of AIRES also possesses internal monitoring procedures that constantly check and record particles reaching ground and/or passing across predetermined observing surfaces located between the ground and injection levels.

From this description, it shows up clearly that the air shower simulation programs consist of various interacting procedures that operate on a data set with a variable number of records, modifying its contests, increasing or decreasing its size accordingly with predetermined rules.

It is necessary to do a modular design of such a program to make it more manageable; and this is particularly relevant for the case of the algorithms related with the physical laws that rule the interactions where –as mentioned– there are still open problems requiring continuous change and testing of procedures.

Figure 1.1 contains a schematic representation of the modular structure of the main simulation programs. Every unit consists of a set of subroutines performing the tasks assigned to the corresponding unit. In general, every unit can be replaced virtually without altering the other ones. In the case of the external interaction models where complete packages developed by other groups are linked to the simulation program via a few interface routines, the modularity acquires particular importance since it makes it possible to easily switch among the various packages available.

The user controls the simulation parameters by means of input directives. The *Input Directive Language* (IDL) is a set of human-readable directives than provides a comfortable environment for task control. After the input data is processed and checked, control is transferred to the program's kernel. During the simulations the particles of the cascade are generated and processed by several packages. The interactions model package contains the "physics" of the problem.

The job control unit is responsible (among other tasks) of updating the *internal dump file (IDF).* This file contains all the relevant internal data used during the simulation, and is the key for system fault tolerant processing since it makes it possible to restart a broken simulation process from the last update of the IDF.

The kernel interacts also with other modules that generate the output data, namely, log, summary, and task summary script files, internal dump file –in either binary or ASCII (portable) format– and compressed output files generated by the monitoring routines and the particle data output unit.

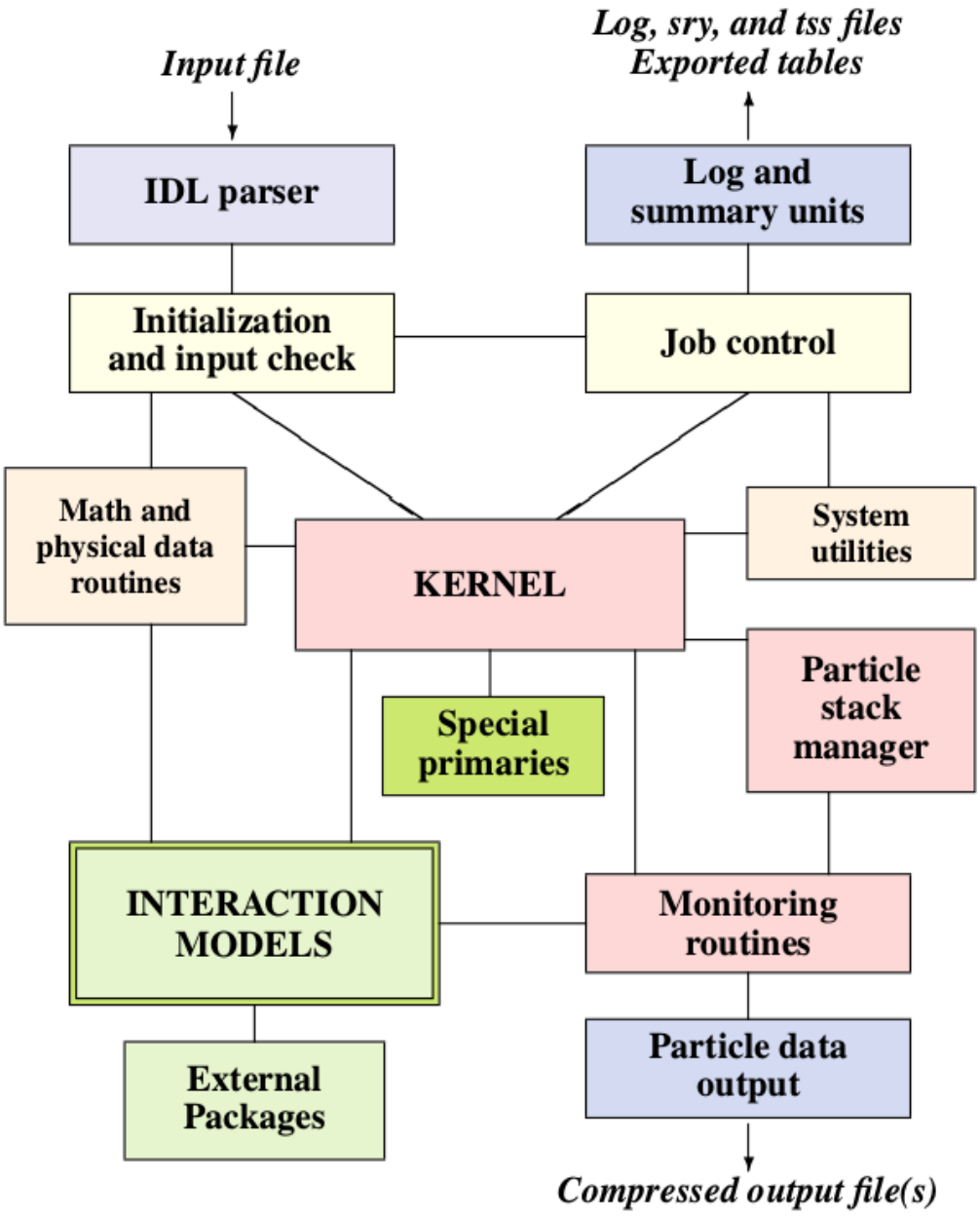


Figure 1.1. *The structure of AIRES main simulation program.*

In the current version of AIRES, there are two compressed output files implemented: The ground particle file and the longitudinal tracking particle file. Records within the ground particle file (longitudinal tracking file) contain data related with particles reaching ground level (passing across observing levels).

Since the number of data records contained in such files can be enormous, a special compression mechanism has been developed to reduce file size requirements. The compressing algorithm is part of the particle data output module. To give an idea of the space needed to store the particle records, let us consider the case of the ground particle file with its default settings: For each particle reaching ground and fulfilling certain (user settable) conditions, a 18 byte long record is written. The record data items are: particle identity, statistical weight, position, time of arrival and direction of motion. Leading and trailing records are written before and after an individual shower is completely simulated. Considering, for instance, a "hard" simulation regime where 2×10^{19} eV primary energy showers (proton or iron) are simulated with 10^{-7} relative thinning level using the standard Hillas algorithm (see section 2.3), generate a compressed ground particle file of size less than 11 MB/shower when storing all the particles whose distance from the shower core is larger than 50 m and less than 12 km.

The green unit named "special primaries" consists basically in a kernel-operated interface with user-provided external modules capable of generating lists of particles that will be used to initiate a shower. This feature allows the user to start showers initiated by non-conventional (exotic) primary particles like neutrinos, for example.

The math and physical data routines are called from several units within the program and provide many utility calculations. In particular, they contain the atmospheric model (used to account for the varying density of the Earth's atmosphere) and the geomagnetic field auxiliary routines that can evaluate the geomagnetic field in any place around the world.

1.2 Getting and installing AIRES

AIRES is distributed worldwide as "free software" for all scientists working in educational/research non-profit institutions. Users from commercial or non-educational institutions must obtain the author's written permission before using the software.

The present version of AIRES (19.04.08) can be obtained from the World Wide Web, at the following site:

`aires.fisica.unlp.edu.ar`

AIRES is distributed in the form of compressed UNIX tar files. The installation is automatic for LINUX and Mac OS systems. For other operating systems some adaptive work may be needed. Appendix A (page 101) contains detailed instructions on how to install AIRES and/or maintain an existing installation.

Chapter 2

General characteristics of AIRES

The aim of this chapter is to introduce the basic concepts needed to adequately define the problem being considered.

2.1 The environment of an air shower

2.1.1 Coordinate system

The AIRES coordinate system is a Cartesian system whose origin is placed at sea level at a user-specified geographical location. The xy plane is located horizontally at sea level and the positive z-axis points upwards. The x-axis points to the "local" magnetic North, that is, the local direction of the horizontal component of the geomagnetic field (see section 2.1.5 for details). The y-axis points to the West.

Figure 2.1 shows an schematic representation of the coordinate system used by AIRES. The xy plane is tangent to the sea level surface, here taken as a spherical surface of radius $R_e = 6371007$ m [14] centered at the Earth's center. The *ground level,* and the *injection level,* refer to spherical surfaces concentric to the sea level surface and intersecting the z-axis at $z = z_g$ ($z_g \geq 0$) and $z = z_i$ ($z_i > z_g$) respectively.

inputcoord1fig.tex

Figure 2.1. *AIRES coordinate system.*

The *shower axis* of a shower with zenith angle Θ is defined as the straight line that passes by the intersection point between the ground level and the z-axis, and makes an angle Θ with the z-axis ($0 \leq \Theta < 90^\circ$). The azimuth angle Φ is the angle between the horizontal projection of the shower axis and the x-axis ($0 \leq \Phi < 360^\circ$).

In AIRES version 1.2.0, all the spherical surfaces mentioned in the preceding paragraphs were approximated as planes. This approximation is justified every time the horizontal distances involved are negligible in comparison with the Earth's radius, R_e. This is the case for showers whose zenith

angle is small, but certainly not for those with large zenith angles, especially for quasi-horizontal showers.

For AIRES version 1.4.0 or later the curvature of the Earth is taken into account to make it possible to reliably simulate showers with zenith angles in the full range $0 \leq \Theta < 90^\circ$. Since full spherical calculations are computationally expensive, an effort was made to optimize the corresponding algorithms. These optimizations are based on two key concepts: (i) Even if a non-vertical shower can start in a very distant point, most of the shower development takes place relatively near the z-axis where the "plane Earth" approximation is acceptable. (ii) Many calculations that employ spherical geometry can be substantially simplified if the coordinate system is temporarily rotated so the involved point lies near the new z-axis, and plane geometry is used in the rotated system. If necessary, an inverse rotation is applied to express results in the original coordinate system.

In order to apply the first concept, a zone where the Earth can be acceptably approximated as plane must be defined. As it will be justified later in this chapter (see section 2.1.4), the Earth's spherical shape can be ignored in a conic region region centered at the z-axis, with a varying diameter ranging from 8 km at sea level to 45 km at an altitude of 100 km.a.s.l. The average limits of that region (about 22 km diameter) are indicated in figure 2.1.

To fastly perform the rotation operations needed to express coordinates and vector in a temporary local coordinate system, it results convenient to use a redundant set of coordinates, defined as follows: Let $\mathbf{r}$ be the position vector of a point with coordinates (x, y, z). We define the *vertical altitude,* z_v, of the point as the minimum distance between the point and the sea level surface. It is straightforward to demonstrate that

$$(R_e + z_v)^2 = (R_e + z_c)^2 + \rho^2 \tag{2.1}$$

where $\rho^2 = x^2 + y^2$ and $z_c = z$ denotes the point's *central altitude,* an alternative way to express the z-coordinate which stresses the fact that this coordinate is always measured along the same central axis. The redundant set of coordinates

$$(x, y, z_c, z_v) \tag{2.2}$$

is used by AIRES to define the position of a point. The difference between z_c and z_v gives information about how far from the z-axis is the point, and in the "plane Earth" zone z_v is set equal to z_c.

This way of taking into account the Earth's shape in the simulations proved to be accurate enough when compared with exact procedures while being economic from the computational point of view.

2.1.2 Atmosphere

The Earth's atmosphere is the medium where the particles of the shower propagate and their evolution depends strongly on its characteristics. The simulations must therefore be based on realistic models of the relevant atmospheric quantities.

The atmosphere has been extensively measured and studied during the last decades. As a result, a variety of models and parameterizations of measured data have been published. Among them, the

so-called *US standard atmosphere* [19] is a widely used model based on experimental data[1]. We have selected it as a convenient default model to use in AIRES which gives an acceptably realistic approximation of the average atmosphere.

Besides the default model, AIRES accepts other atmospheric models, including the possibility of adding user-defined custom models, as explained with more detail in section ....

An evident characteristic of the atmospheric medium is that of being inhomogeneous. Its density, for instance, diminishes six orders of magnitude when the altitude above sea level passes from zero to 100 km, and another additional six orders for the range 100 km to 300 km [20]. This fact is taken into account in the model we have selected, where most of the relevant observables are regarded as functions of the altitude above sea level, or vertical altitude, h: The atmosphere is thus a spherically symmetric "layer" a few hundreds kilometers thick, whose internal radius is the Earth's radius (6370 km).

For a variety of processes that the particles can undergo during the development of the shower, it is essential to know the chemical composition as well as the density of the medium they are passing through [21]. For this reason, we have studied the behavior of these two quantities, especially their dependence with the vertical altitude.

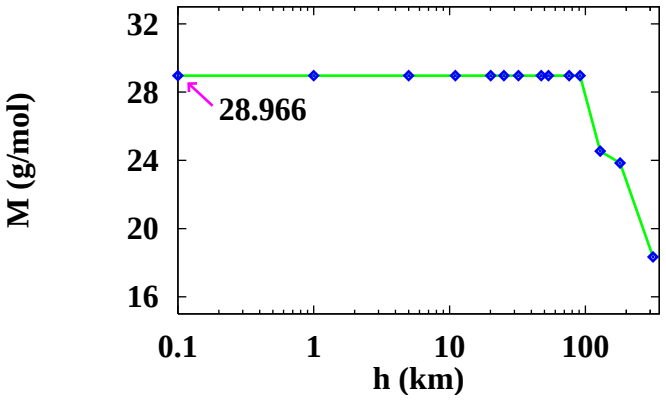


Figure 2.2. *Mean molecular weight of the atmosphere as a function of the vertical altitude (US standard atmosphere [20]). The line is only to guide the eye.*

The chemical composition of the air, as given by the mean molecular weight, remains virtually unchanged in all the region $0 \leq h \leq 90$ km, and diminishes progressively for larger values of h. This clearly shows up in figure 2.2, where the US standard atmosphere mean molecular weight [20] has been plotted versus the vertical altitude. The constant value $M = 28.966$ is the mean molecular weight corresponding to an atomic mixture of 78.47% N, 21.05% O, 0.47% Ar and 0.03% other elements. The corresponding mean atomic weight (atomic number) is 14.555 (7.265). The ratio between mean atomic number and weight is 0.499.

On the other hand, the density of the air does change considerably with the vertical altitude, as shown in figure 2.3. The dots are the US standard atmosphere data, taken from reference [20]. The

[1]The US standard atmosphere is sometimes referred as the US extension of the ICAO (International Civil Aviation Organization) standard atmosphere.

green full line corresponds to Linsley's parameterization of the US standard atmosphere [22], also called Linsley's atmospheric model or Linsley's model, which effectively reproduces very accurately the US standard atmosphere data. The isothermal atmosphere

$$\rho(h) = \rho_0\, e^{-gMh/RT} \tag{2.3}$$

was also plotted (dotted red line) for comparison. ρ_0 and T match the corresponding US standard atmosphere values at sea level.

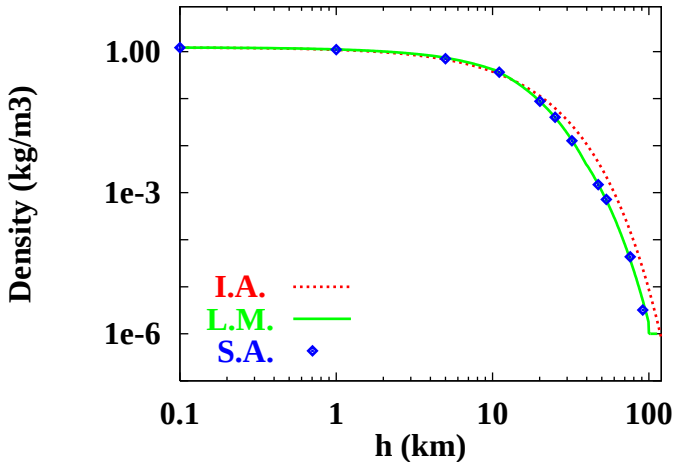


Figure 2.3. *Density of the air as a function of the vertical altitude. The dots represent the US standard atmosphere data [20], while the full green line corresponds to Linsley's model [22] and the dashed red one to the isothermal atmosphere* $\rho(h) = \rho_0\, e^{-gMh/RT}$ *with* $\rho_0 = 1.225\ \mathrm{kg/m^3}$, $M = 28.966$ *and* $T = 288\ K$.

It is worthwhile mentioning that Linsley's model is limited to altitudes less than 100 km. Currently, in AIRES the model has been extended up to $h_{\max} \sim 420$ km, approximately; the density is considered to be zero for $h > h_{\max}$. This approximation helps very much to simplify different algorithms used in air shower simulations while being absolutely justified since only affects an atmospheric zone placed much above the region where the air showers take place, which at most extends up to 50 vertical kilometers above sea level.

For the same reason, the chemical composition of the air can be assumed to be constant in the full range of non-vanishing density ($0 \le h \le h_{\max}$). As shown in figure 2.2, this only affect the very upper layer of the atmosphere, with altitudes larger than 90 km.

A further approximation that will be made when necessary is to assume that the air is a "pure" substance made with "air" atoms whose nuclei possess charge Z_{eff} and mass number A_{eff}. To match the actual molecular weight, it is necessary to set $Z_{\mathrm{eff}} = 7.3$ and $\langle Z_{\mathrm{eff}}/A_{\mathrm{eff}}\rangle = 0.5$ [1].

The density of the air is not directly used by the related algorithms: The quantity that naturally describes the varying density of the atmospheric medium is the so called *vertical atmospheric depth,* X_v, defined as follows:

$$X_v(h) = \int_h^\infty \rho(z)\, dz. \tag{2.4}$$

The integration path is the vertical line that goes from the given altitude, h, up to infinity. The usual unit to express X_v is $\mathrm{g/cm^2}$. In figures 2.4 and 2.5, $X_v(h)$ (Linsley's model) is plotted against h. Notice that $X_v(0) \approx 1000\ \mathrm{g/cm^2}$ and $X_v(h) \to 0$ for $h \to \infty$ as expected.

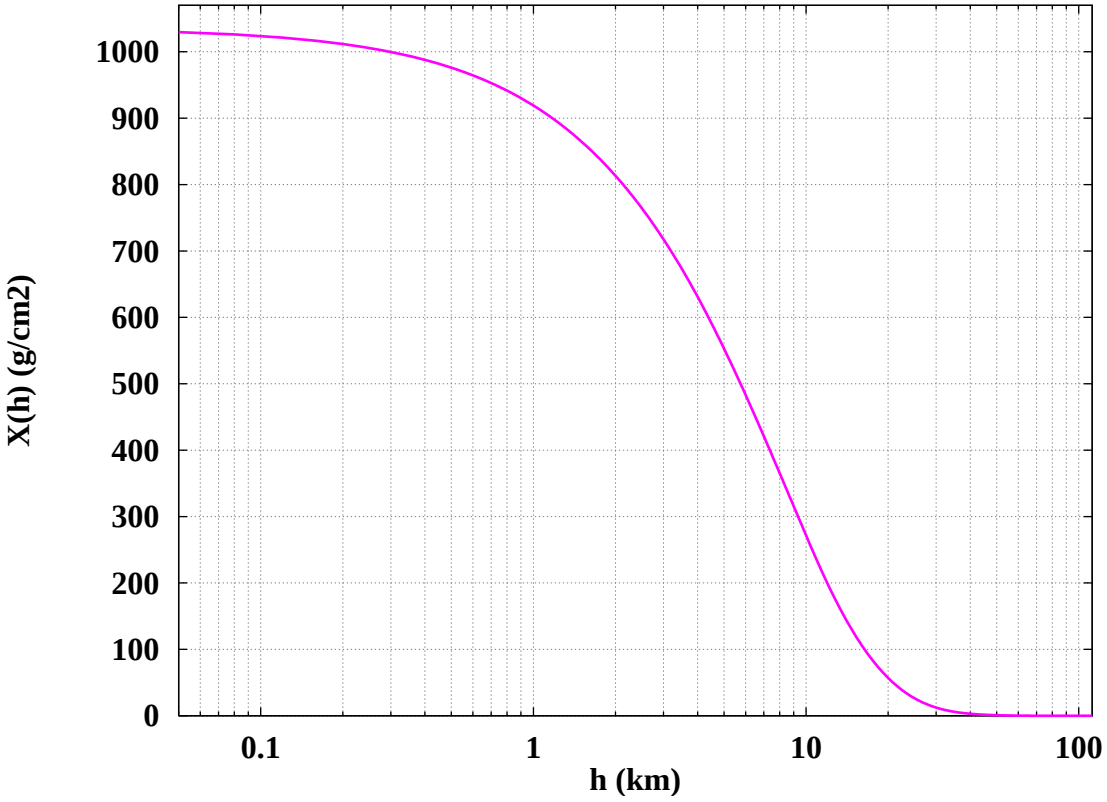


Figure 2.4. *Vertical atmospheric depth, X_v, versus vertical altitude over sea level, h, accordingly with Linsley's model [22].*

$\rho(h)$ can be obtained from $X_v(h)$ via

$$\rho(h) = -\frac{dX_v(h)}{dh}. \tag{2.5}$$

Linsley's parameterization of $X_v(h)$ [22], is done as follows: (i) The atmosphere is divided in L layers. For $l = 1, \ldots, L$ layer l starts (ends) at altitude h_l (h_{l+1}). It is clear that $h_1 = 0$ and $h_{L+1} = h_{\max}$. (ii) $X_v(h)$ is given by:

$$X_v(h) = \begin{cases} a_l + b_l e^{-h/c_l} & h_l \leq h < h_{l+1}, \quad l = 1, \ldots, L-1 \\ a_L - b_L(h/c_L) & h_L \leq h < h_{L+1} \\ 0 & h \geq h_{L+1}. \end{cases} \tag{2.6}$$

Where the coefficients a_l, b_l and c_l, $l = 1, \ldots, L$ are adjusted to fit the corresponding experimental data. The coefficients used in AIRES, which correspond to a model with $L = 5$ layers, are listed in table 2.1, and are the ones that come out from a fit to the US standard atmosphere data. The Linsley's model prediction for $\rho(h)$ plotted in figure 2.3 was obtained using this coefficient set and equations (2.6) and (2.5).

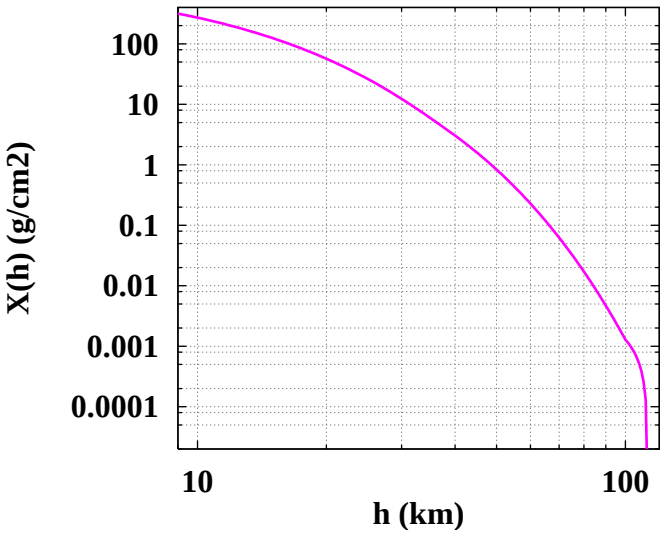


Figure 2.5. *Same as figure 2.4, but for altitudes larger than 10 km.*

Another important property of Linsley's parameterization is that $X_v(h)$ can easily be inverted to obtain $h = X_v^{-1}(X)$ $(X > 0)$: Let $X_l = X_v(h_l)$, $l = 1, \ldots, L$, then

$$h = \begin{cases} -c_l \ln\left(\dfrac{X - a_l}{b_l}\right) & X_{l+1} < X \leq X_l, \quad l = 1, \ldots, L-1 \\ c_L(a_L - X)/b_L & 0 < X \leq X_L, \end{cases} \tag{2.7}$$

where the replacement $X_v(h_{L+1}) = 0$ has been made.

A quantity related to the vertical depth that appears frequently in air shower calculations is the *slant atmospheric depth, X_s*, defined similarly as X_v (equation (2.4)) but using a non-vertical integration path. In most applications the integration path is a straight line going along the shower axis, from the given point to infinity. In this case X_s takes the form:

$$X_s(z) = \int_z^{\prime\,\infty} \rho(z_v)\, dl, \tag{2.8}$$

where the prime in the integral indicates that the path is along a non-vertical line and z_v is the vertical altitude defined in equation (2.1).

The integral in equation (2.8) cannot be solved analytically in the general case of an arbitrary geometry (see page 214). If the Earth's curvature is not taken into account (plane Earth), then it is straightforward to prove that

$$X_s(h) = \frac{X_v(h)}{\cos\Theta}, \tag{2.9}$$

where Θ is the zenith angle of the shower axis (see section 2.1.1). From this equation it comes out that X_s depends not only on h but also on Θ and the location of the ground surface.

Layer l	***Layer limits*** (km) ***From***	***To***	a_l (g/cm^2)	b_l (g/cm^2)	c_l (m)
1	0	4	−186.5562	1222.6562	9941.8638
2	4	10	−94.9199	1144.9069	8781.5355
3	10	40	0.61289	1305.5948	6361.4304
4	40	100	0.0	540.1778	7721.7016
5	100	∼113	0.01128292	1	10^7

Table 2.1. *Linsley's model coefficients for the US standard atmosphere [22]. The number of layers is* $L = 5$.

Unless otherwise specified, any reference to atmospheric depth, or depth, is assumed to be a reference to X_v which may also be noted simply X.[2]

2.1.3 The slant depth and the Earth's curvature.

Many air shower observables, especially the ground level distributions, depend on the thickness of the air layer that separates the starting point of an air shower from the ground level. For non-vertical showers starting at the top of the atmosphere, this thickness is measured in terms of the slant depth evaluated at ground level, $X_s(z_g)$. The plane Earth approximation given by equation (2.9) is usually employed to evaluate that quantity. However, this approximate equation can give inaccurate estimations for large zenith angles, and in fact it is divergent for $\Theta = 90°$.

Zenith angle (deg)	***Curved Earth***		***Plane Earth***	
	length (km)	***path*** (g/cm^2)	***length*** (km)	***path*** (g/cm^2)
0	110	1036.1	110	1036.1
30	127	1195.9	127	1196.4
45	154	1463.6	156	1465.3
60	215	2065.1	220	2072.2
70	303	3003.7	322	3029.4
80	518	5765.5	633	5966.7
85	757	10571.7	1262	11887.9
89	1083	25919.3	6303	59367.2
90	1189	36479.9	∞	∞

Table 2.2. *Total shower axis length* (m) *and slant path* (g/cm^2) *measured from the top of the atmosphere (110 km.a.s.l) down to sea level, tabulated versus the zenith angle.*

To precisely estimate $X_s(z_g)$ we have evaluated numerically the integral of equation (2.8) for various representative cases. In table 2.2 the results corresponding to $z_g = 0$ (ground level located

[2]Notice that in some publications the symbol X is used to represent the slant depth.

at sea level) are tabulated for different zenith angles. The top of the atmosphere is located at an altitude of 110 km.a.s.l, and Linsley's parameterization is used in the calculations. The respective data corresponding to the plane Earth model are also tabulated for comparison purposes.

The tabulated quantities indicate that the plane and curved Earth estimations differ in less than 4% for all zenith angles $\Theta \leq 80^\circ$, and the differences increase notably as long as the zenith angle approaches 90°.

The geometrical length of the shower axis, a, is also tabulated for both models. In the plane Earth approximation this length is given by

$$a = \frac{z_{\max} - z_g}{\cos\Theta}, \tag{2.10}$$

where $z_{\max}$ is the (vertical) altitude of the top of the atmosphere (110 km in the present case). On the other hand, if the Earth's curvature is taken into account, the expression for a becomes

$$a = \sqrt{(R_e + z_{v\max})^2 - (R_e + z_g)^2 \sin^2\Theta} - (R_e + z_g)\cos\Theta, \tag{2.11}$$

where $z_{v\max}$ stands for the vertical altitude of the injection point (110 km). Equation (2.10) is the $R_e \to \infty$ limit of equation (2.11).

2.1.4 Range of validity of the "plane Earth" approximation

In section 2.1.1 (page 9) it is specified that the limit of the "plane Earth" zone is located at a certain distance from the central z-axis. This distance varies linearly with the altitude and goes from 4 km at sea level up to 22.5 km at 100 km above sea level.

To determine the boundaries of that zone, that is, a region where plane geometry can safely be used in the involved procedures, the requirement of expressing the vertical depth of a given point with enough precision was taken into account. The condition actually imposed can be defined in the following terms: Let d be the horizontal distance of a certain point to the z-axis, and let z and z_v be the point's central and vertical altitudes. Let

$$\Delta X(d) = X_v(z_v) - X_v(z). \tag{2.12}$$

In a plane geometry, ΔX is zero for all d provided z is kept fixed. We can use this quantity to determine a safe "plane zone" imposing a bound on ΔX. After a series of technical considerations, too many to be explained in detail here, we concluded that the geometry can be acceptably taken as plane for all points whose distances to the z-axis are less than $d_{\max}$ defined by the condition[3]:

$$\Delta X(d_{\max}) < 0.25\ \mathrm{g/cm^2} \quad \textbf{AND} \quad 2\,\Delta X(d_{\max}) < 1\,\% \times X_v(z). \tag{2.13}$$

Using equations (2.1) and (2.13), and taking into account that $\Delta X \cong (z_v - z)\rho(z)$, it is simple to obtain estimations for $d_{\max}$ at different altitudes. At sea level, for example, where the vertical depth is approximately 1030 $\mathrm{g/cm^2}$, and the density of the air is 1.22 $\mathrm{kg/m^3}$, we obtain

$$d_{\max} \lesssim 5.5\ \mathrm{km}. \tag{2.14}$$

[3]This requirement is more stringent that the one used for AIRES version 1.4.2a or earlier. The original equations [18] were not adequate in certain particular conditions, namely, quasi-horizontal showers, and were thus modified.

The same calculation for 100 km above sea level yields

$$d_{\max} \lesssim 24 \text{ km}. \tag{2.15}$$

The boundaries of used by AIRES (see section 2.1.1) agree with these results.

2.1.5 Geomagnetic field

All charged particles that move near the Earth are deflected by the geomagnetic field. Such deflections are taken into account in the internal algorithms of AIRES.

The Earth's magnetic field, $\mathbf{B}$, is described by its strength, F, $\mathrm{F} = \|\mathbf{B}\|$; its inclination, I, defined as the angle between the local horizontal plane and the field vector; and its declination, D, defined as the angle between the horizontal component of $\mathbf{B}$, H, and the geographical North (direction of the local meridian). The angle I is positive when $\mathbf{B}$ points downwards and D is positive when H is inclined towards the East.

Let (B_x, B_y, B_z) be the Cartesian components of $\mathbf{B}$ *with respect to the AIRES coordinate system* (section 2.1.1). They can be obtained from the field's strength and inclination via

$$B_x = F\cos\mathrm{I}, \quad B_y = 0, \quad B_z = -F\sin\mathrm{I}. \tag{2.16}$$

B_y is always zero by construction, since in the AIRES coordinate system the x-axis points to the local *magnetic* north, defined as the direction of the H component of the geomagnetic field.

There are two alternatives for specifying the geomagnetic field in AIRES: (i) Manually, entering F, I and D. (ii) Giving the geographical coordinates, altitude and date of a given event. In the later case, the magnetic field is evaluated using the International Geomagnetic Reference Field (IGRF) [15], a widely used model based on experimental data that gives accurate estimations of all the components of the Earth's magnetic field.

We are not going to place here any further analysis of the geomagnetic field and its implementation in an air shower simulation program. The interested reader can consult reference [23] which contains a detailed description of general aspects of the geomagnetic field and the IGRF, together with a discussion about the practical implementation of the deflection procedure and an analysis of the effect of the geomagnetic field on several air shower observables.

2.2 Air showers and particle physics

We are going to describe here how the particles of an air shower are identified and processed and which interactions are taken into account.

2.2.1 Particle codes.

AIRES recognizes all the particles commonly taken into account in air shower simulations plus additional ones included for completeness. Each particle is internally identified by a particle code. It

is important to notice, however, that user level particle specifications are made by means of *particle names* instead of numeric codes.

Table 2.3 lists AIRES particle codes, together with the corresponding particle names and synonyms.

Nuclear codes are set taking into account Z (atomic number), N (number of neutrons) and $A = Z + N$ (mass number), in a computationally convenient codification formula:

$$\text{code} = 100 + 32\,Z + (N - Z + 8), \tag{2.17}$$

with $0 \leq N - Z + 8 \leq 31$. Taking $1 \leq Z \leq 26$ (from hydrogen to iron), this coding system allows to uniquely identify all known isotopes.

Regarding the names of nuclei, they can be specified in several ways: (i) By their chemical names, for example **Feˆ56** (56 refers to the mass number A, which defaults to the most abundant isotope's mass number when not specified). (ii) By special names, as **Deuterium** for H^2 or **Iron** for Fe^{56}. (iii) By direct specification of Z, N and/or A, for example **NZ 2 2** (He^4), **ZA 26 54** (Fe^{54}), etc.

In certain cases it may be needed to refer to *groups* of particles having some properties in common. There are several particle groups defined in the AIRES system which can be useful in such situations. The most important groups of particles are listed in table 2.4.

2.2.2 Interactions taken into account in the current version of AIRES

The processes which are most relevant from the probabilistic point of view are taken into account in AIRES. In the current version (19.04.08), the following interactions are included:

- **Electrodynamical processes:**
  - Pair production and e^+e^- annihilation.
  - Bremsstrahlung (electrons and positrons).
  - Muon bremsstrahlung and muonic pair production [24].
  - Emission of "knock-on" electrons (δ rays).
  - Compton and photoelectric effects.
  - LPM effect and dielectric suppression.
- **Hadronic processes:**
  - Inelastic collisions hadron-nucleus.
  - Photonuclear reactions.
  - Nuclear fragmentation, elastic and inelastic.
- **Unstable particle decays.**
- **Particle propagation:**

Particle	*Code*	*Name and synonyms*	*Particle*	*Code*	*Name and synonyms*
γ	1	Gamma gamma	n	30	n Neutron neutron
e^+	2	e+ Positron positron	$\bar{n}$	−30	nbar AntiNeutron antineutron
e^-	−2	e− Electron electron	p	31	p Proton proton
μ^+	3	mu+ Muon+ muon+	$\bar{p}$	−31	pbar AntiProton antiproton
μ^-	−3	mu− Muon− muon−	Λ	40	Lambda
τ^+	4	tau+	$\bar{\Lambda}$	−40	Lambdab
τ^-	−4	tau−	Σ^0	41	Sigma0
ν_e	6	nu(e)	$\bar{\Sigma}^0$	−41	Sigma0b
$\bar{\nu}_e$	−6	nubar(e)	Σ^+	42	Sigma+
ν_μ	7	nu(m)	$\bar{\Sigma}^+$	−42	Sigma+b
$\bar{\nu}_\mu$	−7	nubar(m)	$\bar{\Sigma}^-$	43	Sigma−b
ν_τ	8	nu(t)	Σ^-	−43	Sigma−
$\bar{\nu}_\tau$	−8	nubar(t)	Ξ^0	44	Xi0
π^0	10	pi0	$\bar{\Xi}^0$	−44	Xi0b
π^+	11	pi+	$\bar{\Xi}^-$	46	Xi−b
π^-	−11	pi−	$\bar{\Omega}^-$	47	Omega−b
K^0_S	12	K0S	Ω^-	−47	Omega−
K^0_L	13	K0L	Λ^+_c	48	Lambdac+
K^+	14	K+	Λ^-_c	−48	Lambdac−
K^-	−14	K−			
D^0	16	D0			
η	15	eta			
$\bar{D}^0$	−16	D0bar			
D^+	17	D+			
D^-	−17	D−			
D^+_s	18	Ds+			
D^-_s	−18	Ds−			
B^0	20	B0			
$\bar{B}^0$	−20	B0bar			
B^+	21	B+			
B^-	−21	B-			
ρ^0	25	rho0			
ρ^+	26	rho+			
ρ^-	−26	rho−			

Table 2.3. *AIRES particle codes and names. The nuclear coding system and nuclear names are explained in the text.*

Group name and synonyms	*Particles in the group*
NoParticles None	Empty group
AllParticles All	Universal group containing all particles
AllCharged	All charged particles, including all nuclei
MassiveNeutral	All non-charged massive particles
Nuclei	All nuclei
Hadrons	All hadrons
Neutrinos	All neutrinos and anti-neutrinos
EM	$\gamma,\ e^+,\ e^-$
e+−	$e^+,\ e^-$
mu+−	$\mu^+,\ \mu^-$
tau+−	$\tau^+,\ \tau^-$
GPion	$\pi^+,\ \pi^-,\ \pi^0$
GChPion	$\pi^+,\ \pi^-$
GKaon	$K^+,\ K^-,\ K^0_S,\ K^0_L$
GChKaon	$K^+,\ K^-$
GRho	$\rho^+,\ \rho^-,\ \rho^0$
GChRho	$\rho^+,\ \rho^-$
nppbar	$n,\ p,\ \bar{p}$
nnbar	$n,\ \bar{n}$
Nucnucbr	$n,\ \bar{n},\ p,\ \bar{p}$

Table 2.4. *AIRES particle groups.*

- Medium energy losses (ionization).
- Coulomb and multiple scattering.

The hadronic inelastic collisions and photonuclear reactions are processed by means of external hadronic interaction models when their energy is above a certain threshold; otherwise they are calculated using an extension of Hillas' splitting algorithm [4, 28].

AIRES includes (optionally) links to the well-known external hadronic interaction packages, EPOS [6], QGSJET [7], and SIBYLL [10].

2.2.3 Processing the interactions

We are going to briefly describe how the different interactions are processed in AIRES. We shall focus in the computational aspects of these procedures; a more detailed description of the physics involved in such processes is going to be published elsewhere [29].

First of all it is necessary to express that this description is a general one: The actual algorithms do include a number of technical details whose complete explanation is beyond the scope of this work, even if their philosophy is concordant with the scheme here presented.

As mentioned below, in AIRES the particles are stored in arrays (stacks) and processed sequentially. Each particle entry consists of different data items containing the different variables that characterize it: Particle code, energy, position, direction of motion, etc.

For the simulation engine, the shower starts when the primary particle is added to the previously empty stack. Then the stack processing loop begins.

Let E, $\mathbf{r}$, t, $\mathbf{u}$ be respectively the kinetic energy, position, time and direction of motion of a given particle identified by its particle code k_p. When this particle is going to be processed it will suffer one of several possible interactions I_i, $i = 1, \ldots, n$, $n \geq 1$. To fix ideas, let us consider the case of a positron. The possible interactions, I_i, are: annihilation, interaction with an atom from the medium and emission of a "knock-on" electron, and emission of a bremsstrahlung photon.

Evaluating the mean free paths

Every interaction I_i is characterized by its *cross section,* σ_i, or, equivalently, by its *mean free path,* λ_i. λ_i and σ_i are connected via:

$$\lambda_i = \frac{m_{\rm air}}{\sigma_i}, \tag{2.18}$$

where $m_{\rm air}$ is the mass of an atom of the medium the particle propagates trough, that is, an average atom of "air" in the case of air showers. The usual units for λ_i are $\mathrm{g/cm^2}$.

The mean free paths do depend on the kind of interaction and on the particle's instantaneous parameters. They can be calculated analytically for certain interactions; in other cases they must be estimated by means of parameterization of experimental data, and this generally requires extrapolations out of the region corresponding to the measurements. A typical example of this situation is the case of the mean free paths for inelastic collisions particle-nucleus, where "particle" can be proton, gamma, other nucleus, etc. Such mean free paths depend on the energy of the projectile particle, and must be calculated for energies well above the maximum energies attainable in collider experiments.

Figure 2.6 contains plots of the mean free paths corresponding to proton-nucleus, pion-nucleus, kaon-nucleus and Fe-nucleus collisions, plotted as a function of the projectile energy. All the alternative sets of mean free paths available in AIRES are displayed.

Selecting the particle's fate

For each interaction i, λ_i represents the mean path (expressed in "quantity of matter", that is, $\mathrm{g/cm^2}$) the particle should move before actually suffering the interaction. To evaluate the actual path to a given interaction, it is necessary to sample the corresponding exponential probability distribution, $P_i(p_i) = \lambda_i^{-1} \exp(-p_i/\lambda_i)$. Let p_i, $i = 1, \ldots, n$ the set of values obtained after sampling the corresponding distributions for all the possible interactions.

The interaction the particle will actually undergo, also called the *fate* of the particle, is then selected: It is the interaction j corresponding to the minimum of the p_i's, that is, $p_j \leq p_i$ for all i.

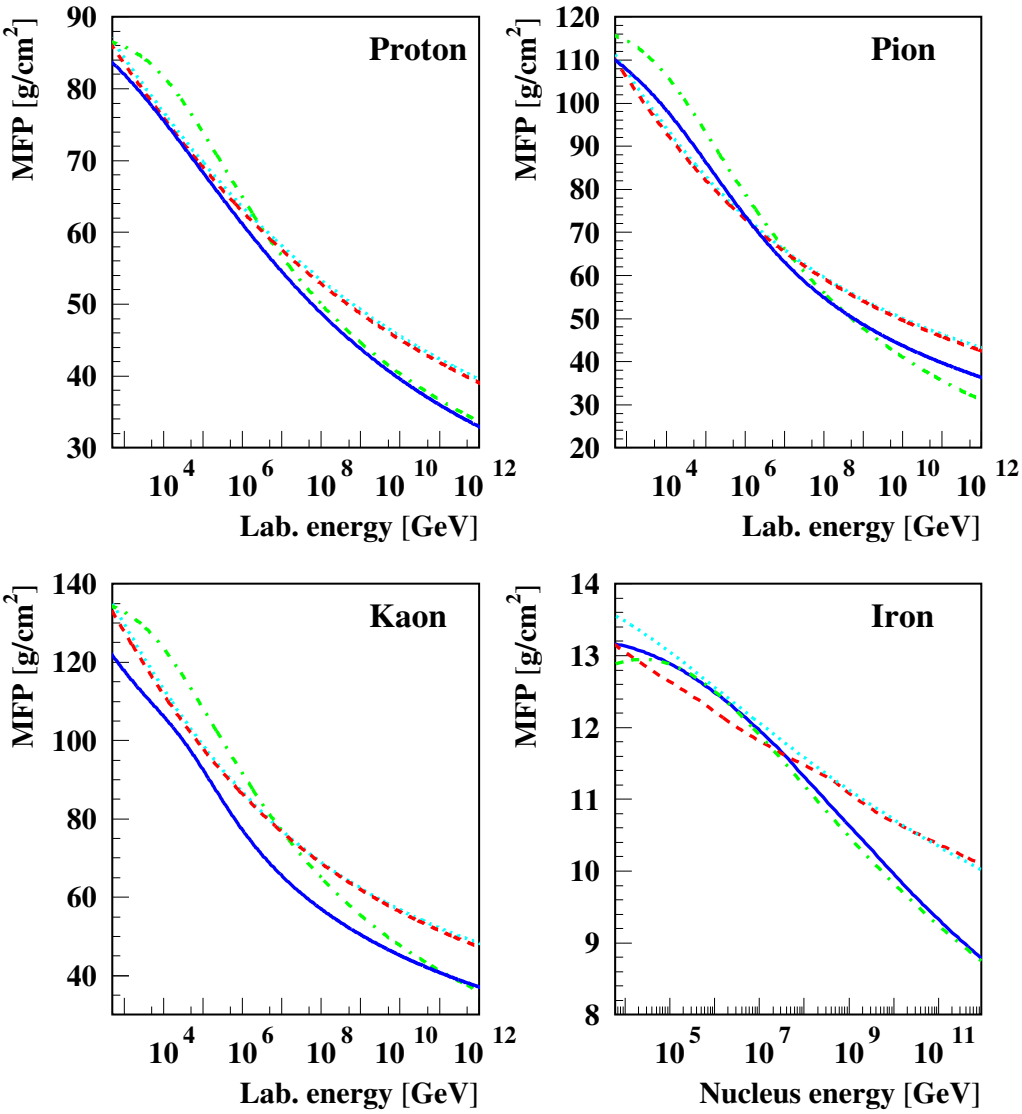


Figure 2.6. *Hadronic mean free paths versus projectile energy (lab system). The solid (blue), and dashed (red) lines represent, respectively the SIBYLL 2.1 and QGSJET01 models. In the proton, pion and kaon cases the mean free paths corresponding to the models SIBYLL 1.6 (dot-dashed, green) and QGSJET99 (dotted, cyan) have been included for comparison. The iron plot includes also the mean free paths evaluated using the AIRES built-in algorithm in the SIBYLL (dot-dashed, green) and QGSJET (dotted, cyan) cases.*

Moving the particle and processing the selected interaction

After the particle's fate has been decided, the corresponding interaction begins to be processed. First, the particle must be advanced the path indicated by p_j. It is necessary to convert the path in a geometrical distance, and this depends on the atmospheric model and the particle's current position. In the case of charged particles, the advancing procedure also takes care of the ionization energy losses, the scattering and the geomagnetic field deflection. During this step, the particle's coordinates, direction of motion and energy can be altered.

The final step is to process the interaction itself. This generally involves the creation of new particles (secondaries) which are added to the corresponding stacks and remain waiting to be processed, and eventually the deletion of the current particle, for example in the case of positron annihilation.

In some cases, it is necessary to apply corrections to the probability distributions used to determine the particle's fate. This happens with processes which have rapidly changing cross sections, or by corrective processes not taken into account in the original selection[4]. The result of the corrective action is that of canceling some interactions. In such cases the particle is left unchanged and remains in the stack for further processing.

Particles arriving to destination

The mechanism so far described is capable of generating and propagating all the secondaries that come after the first interaction of the primary particle. To let the shower finish it is necessary to determine when a particle should no more be tracked. In AIRES this corresponds to the case when one or more of the following conditions hold:

- The particle's energy is below a given threshold (low energy particles)[5].
- The particle's position is out of the interesting region (lost particles).
- The particle reached the ground level.

It is very simple to show that this is enough to ensure that the simulation of a shower will end in a finite time.

Particle monitoring

The simulation programs include several monitoring routines that constantly check the status of the particles being propagated and accumulate data then used to evaluate the different air shower observables.

The events that are monitored are:

[4]The Landau-Pomeranchuk-Migdal (LPM) effect [25, 30, 31] is an example of such kind of processes. The LPM effect implies a reduction of the cross section of $e^{\pm}, \gamma$ processes at very high energies. In AIRES it is implemented as a corrective algorithm whose effect is that of rejecting a fraction of the previously "approved" processes. As a result, the correct cross sections are statistically preserved.

[5]Unstable particles are forced to decays.

- Particles that reach ground level.

- Particles that pass across predetermined observing levels. The observing levels are constant depth surfaces generally located between the injection and ground levels, and separated by a constant depth increment ΔX_o: If N_o is the number of observing levels ($N_o > 1$), and $X_o^{(1)}$ ($X_o^{(N_o)}$) is the vertical depth of the first (last) observing level ($X_o^{(1)} < X_o^{(N_o)}$), then the vertical depth of the other observing levels is given by

$$\begin{aligned} \Delta X_o &= \frac{X_o^{(N_o)} - X_o^{(1)}}{N_o - 1} \\ X_o^{(i)} &= X_o^{(1)} + (i-1)\Delta X_o, \quad i = 1, \ldots, N_o. \end{aligned} \tag{2.19}$$

  Notice that the first observing level is that of highest altitude.

- Charged particles that move across the air. For such particles the continuous energy losses by ionization of the medium are evaluated and recorded.

The data collected by the monitoring routines are used to evaluate different kind of observables, for example:

Longitudinal development of the shower. Tabular data giving the number and energy of particles crossing each defined observing levels.

Shower Maximum. The data collected for the longitudinal development of all charged particles are used to estimate the *shower maximum,* $X_{\max}$, that is, the vertical depth of the point where the number of charged particles reaches its maximum (see section 4.1.1).

Lateral distributions. Frequency distributions recording the number of particles reaching ground, as a function of their distance to the shower core.

Energy distributions. Energy spectra of the different particles at ground level.

Arrival time distributions. Mean ground level arrival time of different particle kinds as a function of their distance to the shower core.

All the output coming from the monitoring routines is saved in the form of data tables that can be easily retrieved by the user (see chapter 4).

2.2.4 Random number generator

AIRES contains many procedures that require using random numbers, the most important example being the propagating procedures that were described in the preceding paragraphs. Those numbers are adequately generated by means of a built-in pseudorandom number generator [1], whose source code is included within the AIRES distribution.

During the early steps of AIRES development, the random number generator was checked with a series of tests, including uniformity and correlation tests among others. In particular, this pseudo-random number generator passed the very stringent "random walk" and "block" tests described in reference [32].

A more detailed description of the different routines associated with the generation of random numbers can be found in appendix D (page 140).

2.3 Statistical sampling of particles: The thinning algorithm

The number of particles that are produced in an air shower grows significantly when the energy of the primary increases. For ultra high energy primaries that number can be large enough to make it impossible to propagate all the secondaries even if the most powerful computers currently available are used. The total number of particles in a shower initiated by a 10^{20} eV proton primary is approximately 10^{11}, being almost impossible even to store the necessary data for such an amount of particles.

The simulations are made possible thanks to a statistical sampling mechanism which allows to propagate only a small representative fraction of the total number of particles. Statistical weights are assigned to the sampled particles in order to compensate for the rejected ones. At the beginning of the simulation, the shower primary is assigned a weight 1.

At the moment of evaluating averages to obtain the physical observables, each particle entry is weighted with the corresponding statistical weight. For example, the observables coming from the monitoring routines, listed in section 2.2.3, are evaluated taking into account those statistical weights. On the other hand, *unweighted distributions* are simultaneously calculated in the cases of longitudinal, lateral and energy distributions. They are useful to monitor the behavior of the sampling algorithm.

The sampling algorithm used in AIRES is called *thinning algorithm* or simply *thinning*. It is an extension of the thinning algorithm originally introduced by A. M. Hillas [4, 1], and was implemented modularly as a procedure which is independent of the units which manage the physical interactions. The original Hillas algorithm and the AIRES extended thinning algorithm are described in the following sections.

2.3.1 Hillas thinning algorithm

Let us consider the process

$$A \rightarrow B_1\ B_2\ \ldots\ B_n, \quad n \geq 1 \tag{2.20}$$

where a "primary" particle A generates a set of n secondaries $B_1, \ldots, B_n$. Let E_A (E_{B_i}) be the energy of A (B_i), and let $E_{\rm th}$ be a fixed energy called *thinning energy*.

Before incorporating the secondaries to the simulating processes, the energy E_A is compared with $E_{\rm th}$, and then:

- If $E_A \geq E_{\rm th}$, every secondary is analyzed separately, and accepted with probability[6]

$$P_i = \begin{cases} 1 & \text{if } E_{B_i} \geq E_{\rm th} \\ \dfrac{E_{B_i}}{E_{\rm th}} & \text{if } E_{B_i} < E_{\rm th} \end{cases} \tag{2.21}$$

- If $E_A < E_{\rm th}$, that necessarily means that the "primary" comes from a previous thinning operation. In this case **only one** of the n secondaries is conserved. It is selected among all the secondaries with probability

$$P_i = \frac{E_{B_i}}{\sum_{j=1}^{n} E_{B_j}}. \tag{2.22}$$

  This means that once the thinning energy is reached, the number of particles is no more increased.

In both cases the weight of the accepted secondary particles is equal to the weight of particle A multiplied by the inverse of P_i.

The fact that the statistical weights are set with the inverse of the acceptance probabilities ensures an unbiased sampling, that is, all the averages evaluated using the weighted particles will not depend on the thinning energy, and will be identical to the "exact" ones obtained for $E_{\rm th} = 0$. Only the fluctuations are affected by the thinning level: If $E_{\rm th}$ is close to the primary energy, then the thinning process begins early in the shower development, and a low number of samples is obtained, with relatively large and fluctuating weights. On the other hand, low thinning energies lead to larger samples with less statistical fluctuations.

Processing large samples demands more computer time, so lowering the thinning level makes the simulation more expensive from the computational point of view.

2.3.2 AIRES extended thinning algorithm

The thinning algorithm of AIRES (19.04.08) includes an additional feature which has proved to be helpful to diminish statistical weight fluctuations in many cases. This extended algorithm was designed to ensure that all the statistical weights are never larger that a certain positive number $W_r > 1$, specified as an external parameter.

The mechanics of the AIRES extended algorithm can be summarized as follows: Let w_A be the weight of particle A, and $W_y < W_r/2$ be an additional (internal) positive number. Consider the number of secondaries in the process (2.20).

- If $n \leq 3$ then
  - If $w_A > W_y$ or $w_A\, E_A / \min(E_{B_1}, \ldots, E_{B_n}) > W_r$ then *all* the secondaries $B_1, \ldots, B_n$ are kept.

[6] The procedure actually used in AIRES implements this step in a technically different way, but retrieving statistically equivalent results.

  - Otherwise the standard Hillas algorithm is used.

- If $n > 3$ then *the standard Hillas algorithm is always used,* but if the weight of the single selected secondary, w_B, happens to be larger than W_r, then m *copies* of the secondary are kept for further propagation, each one with weight $w'_B = w_B/m$. The integer m is adjusted to ensure that $W_y < w'_B < W_r$.

In the AIRES algorithm $W_y = W_r/8$ and the limit W_r is defined via

$$W_r = A_0\, E_{\rm th}\, W_f. \tag{2.23}$$

where A_0 is a constant equal to 14 GeV^{-1} and W_f is an external parameter which can be controlled by the user and that will be referred as the *statistical weight factor.*

In order to optimize the sampling algorithm, it is advantageous to define different weight limits for different particle types. In AIRES two weight factors are defined, $W_f^{(EM)}$ and $W_f^{(H)}$, respectively used when processing electromagnetic or heavy particles. Parameter $W_f^{(H)}$ is specified indirectly, by means of the user-controlled ratio

$$A_{EH} = \frac{W_f^{(EM)}}{W_f^{(H)}} \tag{2.24}$$

that permits evaluating $W_f^{(H)}$ from $W_f^{(EM)}$.

Notice also that W_r depends on the *absolute* thinning energy $E_{\rm th}$. The constant A_0 was adjusted so that $A_0 E_{\rm th}$ gives approximately the position of the maximum of the all particles weight distribution (see below). If $W_f \to \infty$ the extended algorithm reduces to the standard Hillas procedure.

It is a simple exercise to show that this extended thinning algorithm is *unbiased* while ensuring (by construction) that all the particle weights be smaller than the externally specified maximum value W_r of equation (2.23).

It is worthwhile mentioning that this procedure is *not* equal to the thinning algorithm of Kobal, Filipčič and Zavrtanik [5], even if both algorithms do use the concept of keeping bounded the statistical weights.

2.3.3 How does the thinning affect the simulations?

The effect of the standard thinning on different observables evaluated during the simulations can be seen in figures 2.7-2.9. All these simulations were done using identical initial conditions: 10^{19} eV proton showers with vertical incidence; and considering four different thinning energies, namely, $E_{\rm th}/E_{\rm prim} = 10^{-3}$, 10^{-4}, 10^{-6} and 10^{-7}. In all cases the weight limiting mechanism was disabled.

Figure 2.7 (page 28) corresponds to the longitudinal development of all the charged particles, that is the total number of charged particles crossing the different observing levels, as a function of the observing levels' vertical depth.

The plots in this figure show clearly how the statistical fluctuations diminish systematically as long as the thinning energy is lowered. Compare the plot for 10^{-3} relative thinning with the smooth

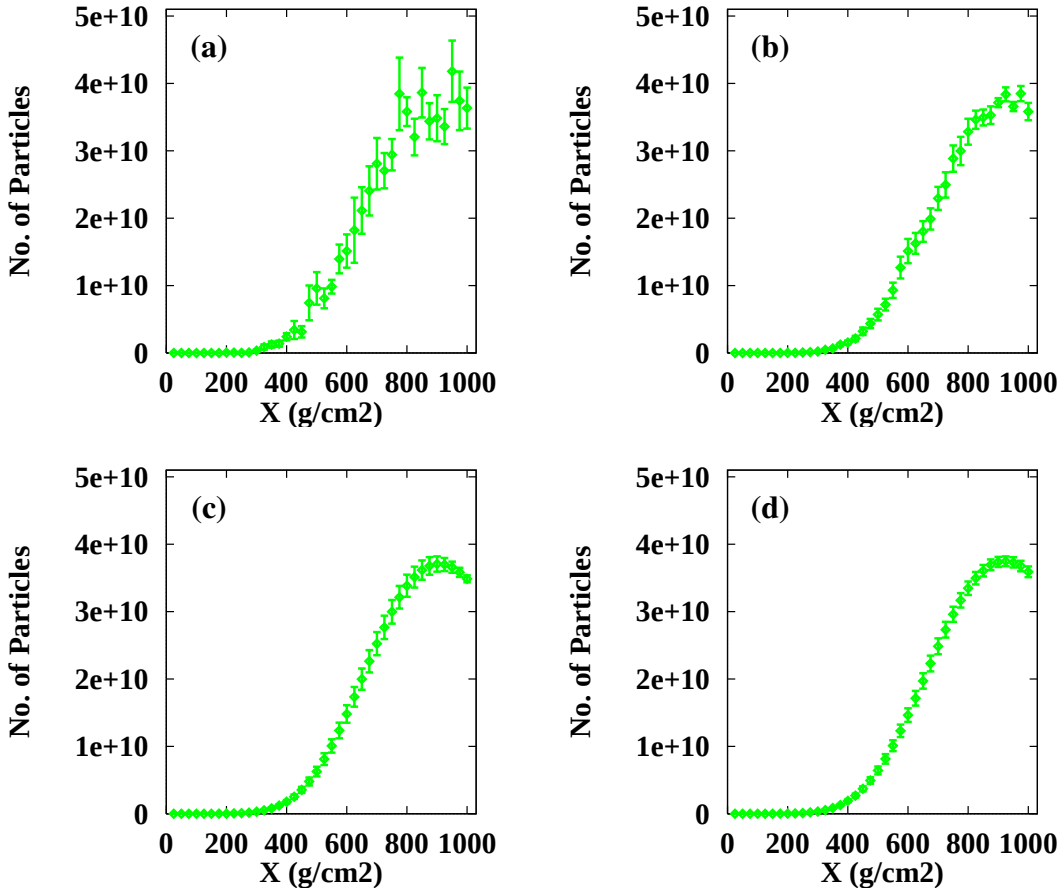


Figure 2.7. *Effect of the thinning energy on the fluctuations of the number of charged particles crossing the different observing levels during the shower development. Ten* 10^{19} *eV vertical proton showers were averaged to obtain the data for each thinning level. The plots labeled (a), (b), (c), (d), correspond to* $E_{\rm th}/E_{\rm prim} = 10^{-3}$, 10^{-4}, 10^{-6} *and* 10^{-7}, *respectively.*

plots obtained for the cases 10^{-6} and/or 10^{-7}. As mentioned, the CPU time required increases each time the thinning energy is lowered. It is interesting to mention that the simulations done at 10^{-7} thinning level required some 6300 times more CPU time than the ones done with 10^{-3} thinning level.

Notice also that the mean positions of the points corresponding to any given depth do not present any evident dependence with the thinning energy, as expected since the Hillas thinning algorithm is an unbiased statistical sampling technique. This observation applies also for the plots of figures 2.8 (page 29) and 2.9 (page 30).

The degree of reduction of the fluctuation does depend on the observable considered. In figure 2.8 (page 29) the lateral distribution of ground electrons and positrons is displayed, again for different thinning levels. It is noticeable the degree of persistence of the noisy fluctuations, which are not completely eliminated even in the 10^{-7} relative thinning case.

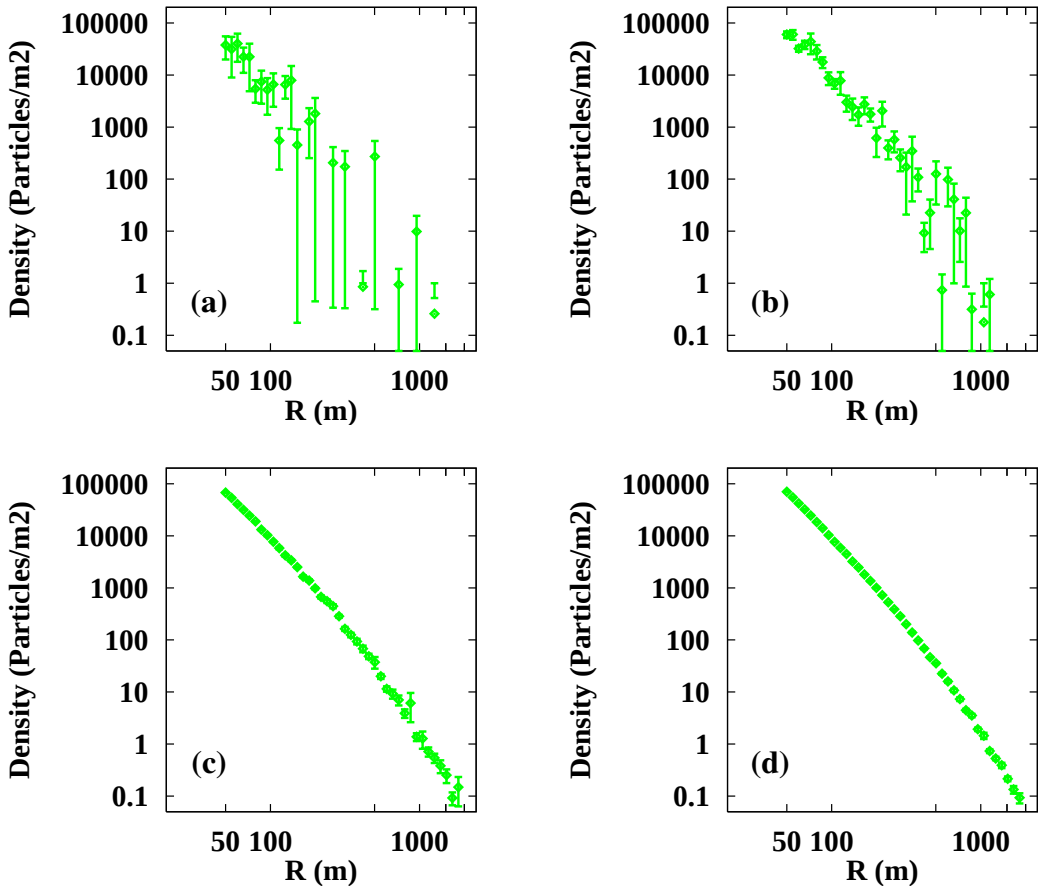


Figure 2.8. *Effect of the thinning energy on the fluctuations of the lateral distribution of electrons and positrons, in the same conditions as in figure 2.7.*

The lateral distribution of muons displayed in figure 2.9 (page 30) reflects another characteristic of the thinning algorithm. Even if the fluctuations are very large for 10^{-3} relative thinning level, they reduce immediately when the thinning is lowered. Compare for example with the plots of figure 2.7 (page 28). To understand the behavior of these distributions it is necessary to recall that the muons are very penetrating particles, that is, they undergo a very reduced number of interactions before reaching ground. Therefore their statistical weights remain small since they are products of a few factors, and this fact is responsible for the low level of fluctuations produced. On the other hand, ground electrons and positrons most likely come out after a long chain of processes involving many predecessor particles, and in such circumstances very large statistical weights are unavoidable, and hence the high level of fluctuations observed in the e^+e^- distribution of figure 2.8 (page 29).

The AIRES extended thinning algorithm can be useful to reduce such kind of fluctuations. To illustrate this point let us consider the sample plots displayed in figure 2.10 (page 31).

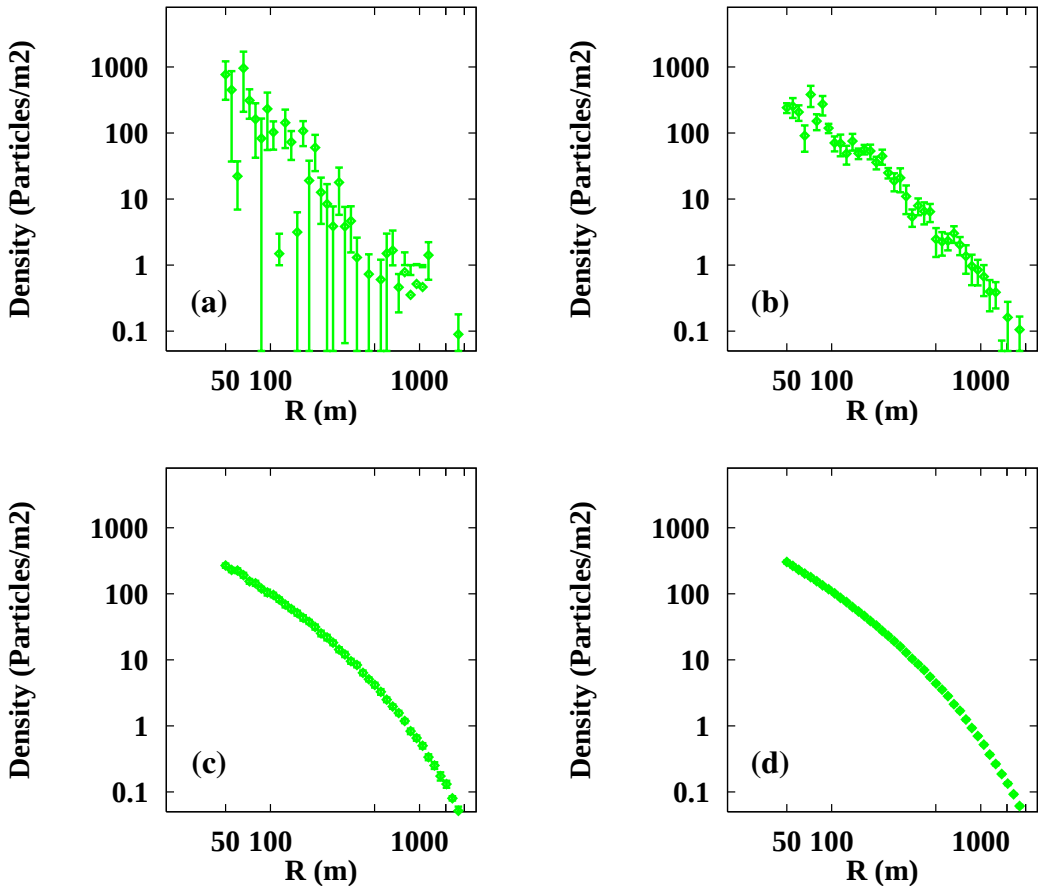


Figure 2.9. *Effect of the thinning energy on the fluctuations of the lateral distribution of muons, in the same conditions as in figure 2.7.*

The outstanding characteristic of these plots is the fact that the density fluctuations diminish when the weight factor ($W_f^{(EM)} = W_f^{(H)} = W_f$) is lowered. In the particular cases of $W_f = 1$ and $W_f = 0.5$ the fluctuations corresponding to the 10^{-5} relative thinning are of the order of the ones corresponding to the 10^{-7} (Hillas algorithm) case (yellow band) which were plotted in all cases for reference.

Looking at the distributions of weights displayed in figure 2.11 (page 32), it is possible to understand the action of the weight limiting mechanism. The distributions labeled "nl" (blue lines) correspond to the Hillas algorithm case (no weight limits). Considering the the distributions of weights for gammas as a typical case, it is evident that there is a small fraction of particles having weights up to three orders of magnitude larger than the most probable ones. This rare cases are generally the cause of many inconvenients that arise when analyzing the data. The plots for finite W_f show clearly that the distributions present a sharp end (corresponding to the value of W_r). In the case $W_f = 1$

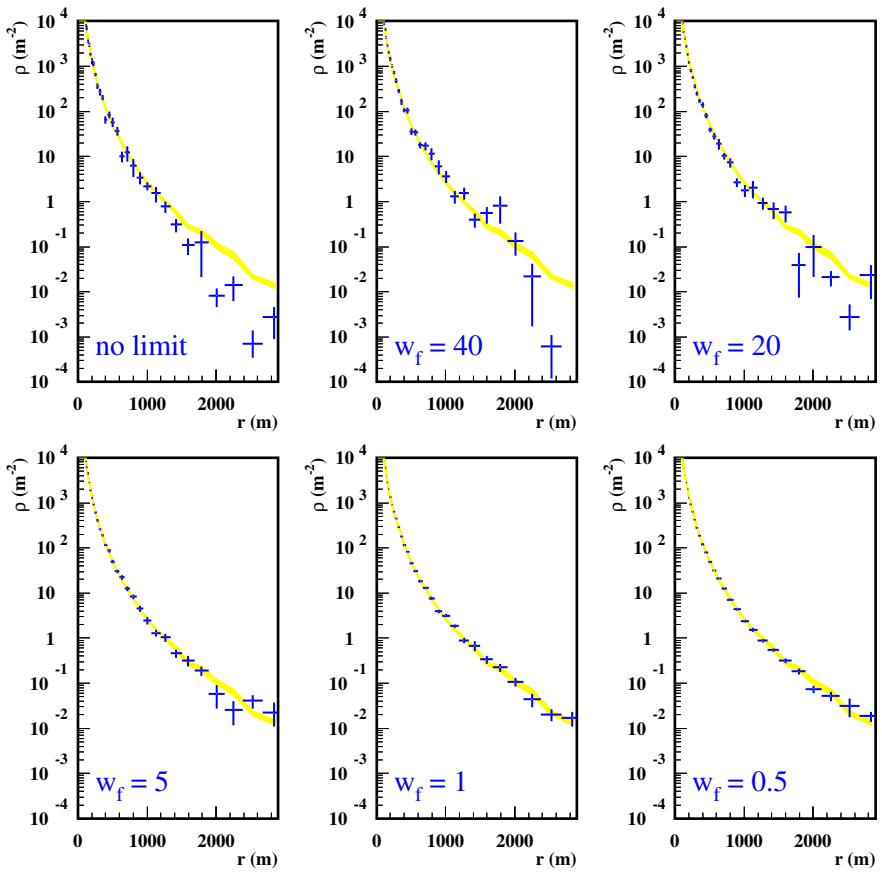


Figure 2.10. *Effect of the AIRES extended thinning on the fluctuations of the lateral distribution of electrons and positrons. The plots correspond to* 10^{19} *eV proton showers simulated with* $E_{\rm th}/E_{\rm prim} = 10^{-5}$*, and different weight factors (*$W_f^{(EM)} = W_f^{(H)} = W_f$*). The yellow bands (▬) correspond to simulations performed in similar conditions, but using the Hillas algorithm at* 10^{-7} *relative level. The width of the bands correspond to the average value plus and minus one RMS error of the mean.*

the gamma distribution ends approximately at the maximum of the "nl" case curve, as expected from equation (2.23), where the factor A_0 is "tuned" to give W_r near the distribution's maximum when W_f is equal to 1.

The muon weights are generally smaller than the electromagnetic counterparts (see the discussion of figure 2.9 in page 29). It is therefore necessary to use a smaller weight limit to modify the corresponding distribution of weights. This is the case of figure 2.11 that corresponds to the case $W_f^{(H)} = W_f^{(EM)}/88$.

It is worthwhile mentioning that the weight distributions corresponding to other thinning energies

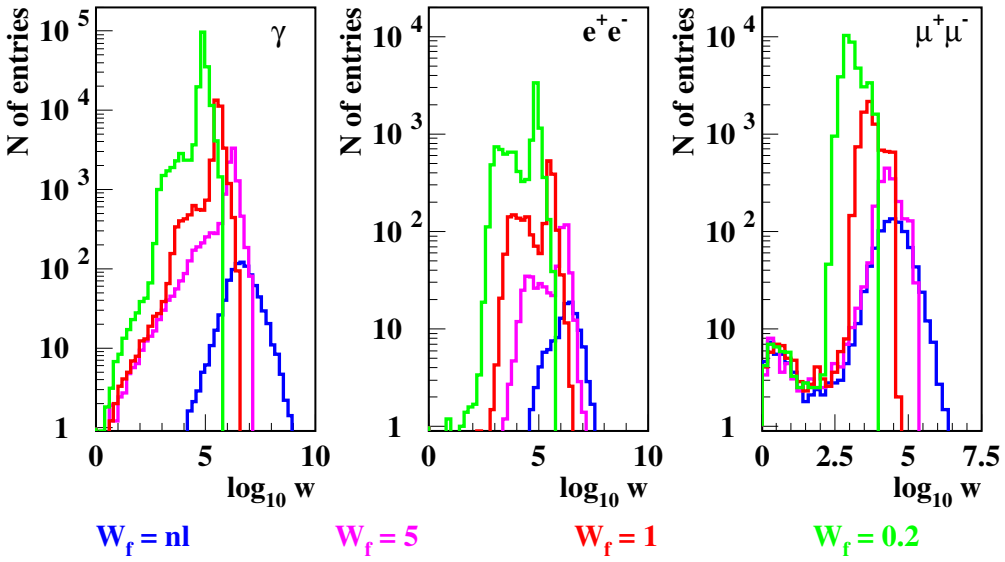


Figure 2.11. *Effect of the AIRES extended thinning on the distribution of weights for different particles (gammas, electrons and positrons, and muons). The plots correspond to* 2×10^{19} *eV proton showers simulated with* $E_{\rm th}/E_{\rm prim} = 10^{-5}$*, and different weight factors* ($W_f^{(EM)} = W_f$, $W_f^{(H)} = W_f^{(EM)}/88$).

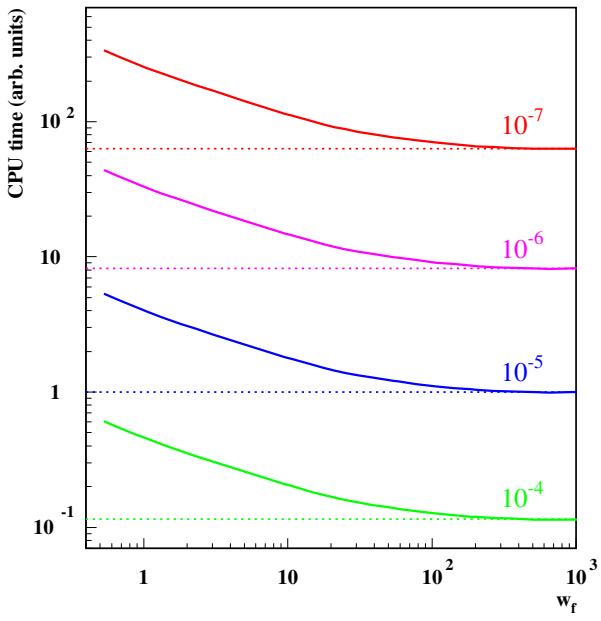


Figure 2.12. *Processor time requirements for the AIRES extended thinning algorithm, plotted versus* $W_f = W_f^{(EM)} = W_f^{(H)}$ *for different relative thinning levels. All cases correspond to* 10^{19} *eV proton showers.*

have the same shape as the ones plotted in figure 2.11 (page 32), but present a global shift in the abscissas scale which is proportional to the logarithm of the thinning level (for example, the weights for the 10^{-6} distribution are one order of magnitude lower than the ones for 10^{-5} and so on).

The improvement in the lateral distribution plots is, of course, not free: The CPU time per shower is increased when W_f decreases. Figure 2.12 (page 32) represents the CPU time consumption per shower as a function of W_f for various thinning energies. The time unit is the average time required to complete a shower simulated with 10^{-5} relative thinning and $W_f \to \infty$.

The CPU time per shower increases monotonically when W_f decreases. For any $E_{\rm th}$ and $W_f = 1$, for example the required time is roughly 5 times larger than the one for $W_f \to \infty$. But it is 1.6 (13) times lower than the one corresponding to the Hillas algorithm for $E_{\rm th}/10$ ($E_{\rm th}/100$). These figures may represent an important time saving factor in certain circumstances, for example when evaluating lateral distributions like the ones of figure 2.10 (page 31).

The use of the AIRES extended thinning algorithm with finite W_f is always recommended, however. Even in the least favorable cases, it is possible to get smoother distributions for every observable setting W_f not larger than 20 and thus eliminating the particle entries with unacceptably large weights.

Chapter 3

Steering the simulations

There are many parameters that must be specified before and during an air shower simulation job. The *Input Directive Language (IDL)* is a part of the AIRES system and consists in some 70 human-readable directives that permit an efficient control of the simulations in a comfortable environment.

The most common IDL directives are described in this chapter, and many illustrative examples are discussed; a detailed description of the IDL language is placed in appendix B (page 106). It is recommended to properly install the software (see appendix A) before proceeding with the following sections.

3.1 Tasks, processes and runs

The simulation of high energy air showers is a CPU intensive task which can demand days of processor time to complete. The AIRES program was designed taking this fact into account: It includes an "auto-saving" mechanism to periodically save into an *internal dump file* (IDF) all relevant simulation data. In case of a system failure, for example, the simulation process can be restarted at the point of the last auto-saving operation, thus avoiding loosing all the previous simulation effort.

The processing block that goes between two consecutive auto-save operations is called a *run.* With *task* we mean a specific simulation job, as defined by the input directives (for example, a task can be "simulate ten proton showers"); and with *process* we identify a system process, which starts when AIRES is invoked and ends when control is returned to the operating system.

A task can be completed after one or more processes, and there can be one or more runs within a process. The limit case consists in having a task finished in a single run (no auto-save) completed in a single process (the program invoked just once).

3.2 The Input Directive Language (IDL)

All the main simulation programs (the default program **Aires** and the executables for the different available external hadronic models **Aires*Model***), and the summary program **AiresSry** read their in-

put directives from the standard input channel, and use a common language to receive the user's instructions. This language is called *Input Directive Language* (IDL).

The IDL directives are written using free format, with one directive per line (there are no "continuation lines", but each line can contain up to 8,000 characters). Special characters like tab characters, for example, are treated as blank characters.

All directives are scanned until either an **End** directive or an end of file is found. Most directives can be placed in any order within the input stream.

IDL directives can be classified as *dynamic* or *static*. Dynamic directives are processed every time the input data is scanned. Static ones can be set only at the beginning of a task: Any subsequent setting will not be taken into account. For instance, the maximum CPU time per run is controlled by a dynamic directive (it can be changed at the beginning of every process, and is a parameter that does not affect the results of the simulation); the ground altitude, instead, is an example of a static parameter that cannot be changed during the simulations.

The IDL sentence begins with the directive name. IDL is a case sensitive language, and in general directive names mix capital and lowercase letters. The directives can be abbreviated. Consider for example the following directive name **PrimaryParticle**. You must specify the underlined part, and may or may not use the remaining characters (**Primary**, **PrimaryPart**, **PrimaryParticle** refer to the same instruction).

3.2.1 A first example

There are four directives that should be always specified before starting a simulation task, namely, the ones that control the task name[1], the statements that provide the primary particle type and energy specifications and the directive which sets the total number of showers to be simulated.

Such a minimal set of specifications can be expressed in terms of IDL directives as follows:

```
Task a_first_example
Primary proton
PrimaryEnergy 150 TeV
TotalShowers 3
End
```

These directives, like most IDL directives, are self-explaining and posses a simple syntax. They can be placed in any order. Notice that the particles are specified by their names and physical quantities like the energy, for example, are entered by means of a number plus a unit.

3.2.2 Errors and input checking

Every IDL directive is checked for correct syntax when it is read in. Additionally, some elemental tests of the values given to the directive's parameters are also made. When an error is detected,

[1] Actually, the **TaskName** directive is not mandatory for a task to start, but its default value **GIVE_ME_A_NAME_PLEASE** produces file names which are rather inconvenient to manage, and so it is strongly recommended to always set the name of a task before proceeding with the simulations.

a message is written to the standard output channel. Directives with errors are generally ignored. Consider the following directive:

```
PrimaryEnergy 100 MeV
```

If processed by AIRES, it will give the following error message

```
EEEE
EEEE  dd/Mmm/yyyy hh:mm:ss. Error message from commandparse
EEEE  Numeric parameter(s) invalid or out of range.
EEEE  >PrimaryEnergy 100 MeV<
EEEE
```

which indicates that the energy specification is out of range.[2]

AIRES diagnostic messages always include a brief explanation about the circumstances that generated the message, together with the name of the routine that originated it. The messages can be classified in four categories, accordingly with their severity: (i) *Informative* messages are used to notify the occurrence of certain events, and are generally associated with successfully concluded operations. (ii) *Warning* messages. Used to put in evidence certain not completely "normal" situations. In general, processing continues normally. (iii) *Error* messages indicate abnormal events like invalid input directives, etc., as illustrated in the previous example. (iv) *Fatal* messages are issued when a serious error takes place; in this case the program stops.

The IDL instruction set includes some directives that allow checking a given input data set. Let us assume that the input directives are saved into a file named **myfile.inp**. Let us consider also that this file contains the instructions of the first example previously considered.

The instruction set:

```
Trace
CheckOnly
Input myfile.inp
End
```

if processed by AIRES will generate an output similar to the following:

```
. . .
0:0002 CheckOnly
0:0003 Input myfile.inp
1:0001 Task a_first_example
1:0002 Primary proton
1:0003 PrimaryEnergy 150 TeV
1:0004 TotalShowers 3
1:0005 End
0:0004 End
. . .
```

[2]The primary energy must be greater than 500 MeV (see page 123).

The **CheckOnly** directive instructs AIRES to normally read and check the input data, and then stop without actually starting any simulations. The input lines placed after the **Trace** statement are echoed to the terminal. and the **Input** directives allows including IDL directives placed in other files. Notice the format used for directive echoing. It includes the line number as well as the file nesting level, starting by zero for the standard input channel. **Input** directives can be nested and permit splitting the input data into separate files. This is most useful for organizing a set of input files including some common directives in a single shared file included by every particular file, etc.

In UNIX environments it is possible to use one of the scripts of the AIRES Runner System to automatically check a given input file. For details see chapter 5 (page 93).

3.2.3 Obtaining online help

The AIRES simulation and/or summary programs accept instructions that permit obtaining information about AIRES IDL instructions. The information that can be retrieved in this way is not extensive but it can be useful to the experienced user as a quick guide.

Invoking AIRES interactively and typing "**?**" will return a list of the names of the IDL directives. "**? ***" will cause the list to include also the *hidden directives*. The prompt "**Aires IDL**>" typed at the terminal indicates that AIRES understands that this session is interactive. The **Help** command is similar to **?** but it will maintain prompting disabled. During an interactive session it is always possible to enable or disabling the prompt by means of the directive **Prompt**.

There are two other kinds of help that can be obtained using the current AIRES version, namely, "**? tables**" and "**? sites**", which display the list of available output data tables (see section 4.1 and/or appendix C), and the currently defined geographical sites (see section 3.3.5), respectively.

The directive **Exit**, which can be abbreviated as **x**, will cause AIRES to stop immediately without any further action –not even completing the IDL instruction scanning phase– and is useful to end interactive help sessions.

3.2.4 Physical units

There are many IDL directives which include one or more specifications corresponding to physical quantities. In most cases these specifications have the format "number + unit", like in the **PrimaryEnergy** specification of section 3.2.1, for instance. "Number" and "unit" are character strings, the first one indicates the decimal numerical value for the quantity being specified, while "unit" represents the unit in which "number" is expressed. The characters used for the unit field resemble the name assigned in the real world to the corresponding unit, e.g. `TeV` for *Tera-electron-volt.*

This feature of the IDL language makes the input files more readable, and diminishes drastically the possibility of errors in the specifications, especially for those quantities whose validity ranges may span many orders of magnitude. In such cases a number of commonly used multiples or sub-multiples of the fundamental unit are surely available.

The complete list of units currently implemented is displayed in table 3.1.

Magnitude	*Units*	*Conversion factors*
Length	**mm**	10^{-3} m
	cm	10^{-2} m
	m	1 m
	km	10^{3} m
	in	0.0254 m
	ft	12 in
	yd	3 ft
	mi	5280 ft
Time	**ns**	10^{-9} s
	sec	1 s
	min	60 s
	hr	3600 s
Energy	**eV**	10^{-9} GeV
	keV	10^{-6} GeV
	MeV	10^{-3} GeV
	GeV	1 GeV
	TeV	10^{3} GeV
	PeV	10^{6} GeV
	EeV	10^{9} GeV
	ZeV	10^{12} GeV
	YeV	10^{15} GeV
	J	6.24×10^{9} GeV

Magnitude	*Units*	*Conversion factors*
Angle	**deg**	1 deg
	rad	$180/\pi$ deg
Atmospheric depth	**g/cm2**	1 g/cm^2
Density	**g/cm3**	1 g/cm^3
	kg/m3	10^{-3} g/cm^3
	kg/dm3	1 g/cm^3
Magnetic field	**nT**	1 nT
	uT	10^{3} nT
	mT	10^{6} nT
	T	10^{9} nT
	uG	10^{-1} nT
	mG	10^{2} nT
	Gs	10^{5} nT
	gm	1 nT
Temperature	**K**	1 K
	C	$T_{\mathrm{C}} + 273.15$
	R	$(5/9)T_{\mathrm{R}}$
	F	$(5/9)(T_{\mathrm{F}} + 459.67)$

Table 3.1. *Physical units accepted within IDL directives. The underlined keywords indicate the units used internally to store the corresponding quantities. Time specifications using* **hr**, **min** *and/or* **sec** *may consist in the combination of more that one field, like in* **3 hr 30 min**, *for example. The magnetic field unit* **T** *(***Gs***) stands for the SI (cgs) unit Tesla (Gauss), while* **gm** *corresponds to* γ *(1* $\gamma = 1$ *nT). The formulas for temperatures in the Celsius (***C***), Rankine (***R***), and Fahrenheit (***F***) scales indicate how to convert a temperature in the respective scale to the absolute Kelvin scale.*

3.2.5 Carrying on

In figure 3.1 a second example of an IDL input data set is displayed. Notice first that IDL instructions can be commented: All the characters following a '**#**' character are ignored.

The **Skip** statement is also useful to place comments and/or introduce plain text in the input files (with no need of single line comment '**#**' characters), as well as to skip a part of the directives without deleting the lines.

Comments and skipped lines are completely ignored: They just appear in the input file. Sometimes this is not convenient, and it may be desirable to save their contents together with the output generated by the simulating program. The **Remark** directive provides a mean to do this. The statement

```
Remark     JUST AN EXAMPLE
```

placed in the example being discussed, instructs AIRES to place the comment 'JUST AN EXAMPLE' together with the output data. There is no limit in the number of remark instructions that may appear inside a given input instruction set. The **Remark** directive possesses another alternative syntax, very useful for multi-line text:

```
Rem &eor
This is the first line of a multi-line remark.
This is the second line
S  p  a  c  e  s   and TABS     will be honored.
. . .
The label &eor marks the end of the remark.
&eor
```

The directives that follow illustrate a very useful feature if the IDL, which is the possibility of defining *global variables*. Such variables can be used as replacement text within the IDL input stream, and/or be passed to output files or external modules called by AIRES. The variables must be defined before they can be used. This can be done by means of the **SetGlobal** directive. The **Import** directive permits to import OS environment variables. Variables can be overwritten, and deleted using the **DelGlobal** directive.

The global variable replacement mechanism allows also to insert (simple) calculations when needed. Let us consider the following example:

```
SetGlobal LogEprimeV 18.5
. . .
PrimaryEnergy {= pow 10 {= subtract {LogEprimeV} 18}} EeV
```

When parsing the **PrimaryEnergy** directive, the expression between brackets will be processed to return the result of $10^{18.5-18} = 3.16227766$, and therefore the primary energy will finally be set to 3.16227766 EeV. The equal (**=**) character at the beginning of the field between backets indicates that what follows is an expression of the form `function` ***args*** that will be evaluated while parsing the directive. The number of arguments can be 0, 1 or 2 according with the function. The available functions are listed in table 3.2

Function	*Args*	*Operation*
ADD	2	$a_1 + a_2$
SUBTRACT	2	$a_1 - a_2$
MULTIPLY	2	$a_1 \times a_2$
DIVIDE	2	a_1/a_2
POWER	2	$a_1^{a_2}$
MIN	2	$\min(a_1, a_2)$
MAX	2	$\max(a_1, a_2)$
MEAN	2	$(a_1 + a_2)/2$
GMEAN	2	$\sqrt{\lvert a_1 a_2 \rvert}$
ABS	1	$\lvert a_1 \rvert$
NEGATIVE	1	$-a_1$
SQRT	1	$\sqrt{a_1}$
EXP	1	e^{a_1}
LOG	1	$\ln(a_1)$
LG10	1	$\log_{10}(a_1)$
SIN	1	$\sin(a_1)$
COS	1	$\cos(a_1)$
SEC	1	$1/\cos(a_1)$
TAN	1	$\tan(a_1)$
ASIN	1	$\arcsin(a_1)$
ACOS	1	$\arccos(a_1)$
ASEC	1	$\arccos(1/a_1)$
ATAN	1	$\arctan(a_1)$
ATAN2	2	$\arctan2(a_1, a_2)$
DEG2RAD	1	$(\pi/180)a_1$
RAD2DEG	1	$(180/\pi)a_1$
PI	0	π
RANDOM	0	uniform random number in $[0, 1)$
XVATH	1	$X_v(a_1)$ atm. depth [g/cm2] at alt a_1 [m]
HATXV	1	$h(a_1)$ alt. [m] at atm. depth [g/cm2]
DENSATH	1	$\rho(a_1)$ density [g/cm3] at alt a_1 [m]
HATDENS	1	$h(a_1)$ alt. [m] at density [g/cm3]
XMAXEST	2	Rough estimation of $X_{\max}$

Table 3.2. *Available IDL mathematical operations. The function names are NOT case sensitive. When there are no ambiguities, they can be abbreviated downto the first three characters.*

```
#
# An example of an AIRES IDL input data set.

Skip &next

The directive "Skip" skips all text until the label &label is
found. Notice that it is not equivalent to a "go to" statement
since it is not possible to skip backwards.

As it can easily be seen, most directive names are self-explaining.
&next

Remark    JUST AN EXAMPLE

# It is possible to define variables that can be used later within
# the input file and/or be passed to output files or special
# primary modules.

SetGlobal MyVariable  This string is associated with the variable.
SetGlobal VRem        Another variable.

Remark {VRem} # This expands to: Remark Another variable.

Import HOME   # Importing OS environmental variables.


# The input directives define a "task". Tasks are identified by
# their task name and (eventually) version. If not defined, the
# version is zero.

Task mytask   # Use "Task mytask 5" to explicitly set task
              # version to 5.

# The following three directives are mandatory (have no default
# values)

TotalShowers      2
PrimaryParticle   Proton
PrimaryEnergy     1.5 PeV
#
# The remaining directives allow controlling many parameters of the
# simulations. The respective parameters will take a default value
# whenever the controlling directive is not present.
#

PrimaryZenAngle   15 deg
Thinning          1.e-4 Relative # Relative or absolute
                                 # specifications allowed.

Ground     1000 g/cm2

# You can freely set the number of observing levels to record the
# shower longitudinal development. You can define up to 510
# observing levels and (optionally) altitude of the highest and
# lowest levels.

ObservingLevels 41 100 g/cm2 900 g/cm2
```

Figure 3.1. *Sample AIRES input.*

```
#
# Threshold energies. Particles are not followed below these
# energies.

GammaCutEnergy     200 KeV
ElectronCutEnergy  200 KeV
MuonCutEnergy        1 MeV
MesonCutEnergy     1.5 MeV  # Pions, Kaons.
NuclCutEnergy      150 MeV  # Nucleons and nuclei.

#
# Some output control statements.
#

# Compressed particle data files related directives.

SaveInFile     lgtpcles  e+ e-
SaveNotInFile  grdpcles  gamma

# Saving the ASCII (portable) version of the IDF file (ADF), after
# finishing the simulations.

ADF   On

#
# No tables are printed or exported if no PrintTables ExportTables
# directives are explicitly used.
#
PrintTable 1291         # Longit. devel. of all charged particles.
PrintTable 1707         # Energy longitudinal development of muons.
PrintTable 2207 Opt d   # Setting some options.
PrintTable 3001 Opt M   # Here too.
#
ExportTable 2793 Opt M  # Exported tables are placed in separate,
ExportTable 5501        # plain text files for further processing
                        # (e. g. plotting).

End # End of input data stream.
```

Figure 3.1. *(continued)*

The input data set of figure 3.1 continues with the **TaskName** directive and the three mandatory directives already introduced in section 3.2.1.

The directives that follow set some characteristics of the showers that are going to be simulated. The **PrimaryZenAngle** directive gives the shower zenith angle, measured as indicated in figure 2.1 (page 9). This directive, and the directive **PrimaryAzimAngle** permit the user to completely control the inclination of the shower axis. They can be used to set this inclination to a fixed value, or to select variable settings selected at random with adequate probability distributions. In this case the alternative syntax of the directives should be used. For a more detailed description see section 3.3.3 (page 51) and/or appendix B (page 106).

The **GroundAltitude** specification indicates the height above sea level of the ground surface (measured vertically). The specification can be a length or a vertical atmospheric depth expressed in $\mathrm{g/cm^2}$ (see page 117). On the other hand the statement

```
ObservingLevels 41 100 g/cm2 900 g/cm2
```

sets the variables N_o, $X_o^{(1)}$ and $X_o^{(N_o)}$ of equation (2.19).

The IDL instructions continue with five directives that fix the cut energies for different particle kinds. Every particle whose kinetic energy falls below the threshold corresponding to its kind will be no more propagated by the simulation program, as explained in section 2.2.3 (page 20).

There are many observables that can be defined and studied to determine the behavior of air showers with given initial conditions. Generally only a small fraction of these observables are of interest for a determined user; and of course, the set of relevant observables do vary with the particular problem being studied.

These somewhat contradictory facts were taken into account when designing AIRES output units, together with an analysis of the output system of existing programs [1, 36]. As a result, the simulation program was provided with two air shower data output units: The *particle data* unit and the *summary* unit.

The particle data unit generates *compressed particle data files* containing detailed information (in a per particle basis) of particles reaching ground or passing across the different observing levels. The other output unit processes data stored in a number of internal tables (or histograms) which were calculated during the simulations and which correspond to *standard* observables like lateral distributions, energy distributions and so on.

The output system will be treated in detail in chapter 4 (page 69). Nevertheless, it is worthwhile mentioning here that there are several IDL directives that permit controlling its behavior.

In our example of figure 3.1 (page 41), the directives **SaveInFile** and **SaveNotInFile** control the kind of particles that are saved in the corresponding compressed files, identified by their extensions (**lgtpcles** and **grdpcles**).

The default action for the file containing record for the particles reaching ground (extension **grdpcles**) is that all particle kinds must be saved. On the other hand, no particles are saved by default in

the longitudinal tracking particle file (extension **lgtpcles**). Therefore, the statements

```
SaveInFile lgtpcles e+ e-
SaveNotInFile grdpcles gamma
```

mean that only electrons and positrons are going to be saved in the longitudinal file, and that all particles but gamma rays are going to be recorded in the ground particle file. The particle kind specifications may include one or more particle or particle group names (see section 2.2.1).

There may be more than one of these statements for each file, and their meaning depends on the order they are placed within the input data stream. As an example, let us consider the following statements:

```
SaveInFile somefile None
SaveInFile somefile Muons
```

They ensure that only muon records will be saved in file[3] **somefile**: The first statement "clears", and the second enables muons. If the order is changed:

```
SaveInFile somefile Muons
SaveInFile somefile None
```

then the result is that **somefile** will be considered disabled because the last **None** specification prevents any particle kind from being saved in the corresponding file.

The logical switch controlled by the instruction **ADF On**, enables the ***portable dump file,*** the portable version of the IDF file.

The summary unit manages more than 300 output data tables that can be selectively included within the output data. Each table is identified by a numerical code, and the directives **PrintTables** and **ExportTables** permit including a table listing within one of the output files, or generating a separate plain text file with the corresponding table, respectively. The complete list of available tables is placed in appendix C (page 132). No tables are exported or "printed" if no **Export** or **Print** directives are included within the input data. Notice also that there are several options that modify the resulting output. Such options control the normalization of histograms, output format, etc. A more detailed discussion on this subject is placed in section 4.1 (page 69).

It is strongly recommended to edit a plain text file containing some IDL directives, run the simulation program and analyze the obtained output. In UNIX environments this can be made by means of the command

Aires < *myfile.inp*

or, alternatively

Aires*Model* < *myfile.inp*

where ***myfile.inp*** is the name of the file containing the IDL directives, and ***Model*** identifies the external hadronic package linked with the simulation program used (see page 3).

[3] **somefile** actually indicates the *extension* of the corresponding file, like **grdpcles** or **lgtpcles** for example.

Input data listing

The output typed at the terminal by any of the simulation programs will be similar to the sample displayed in figure 3.2. Among other data, AIRES standard output includes a listing of the most important input parameters. All the parameters that are not explicitly set will take a default value. When default values are in effect, it is indicated with a **(D)** symbol placed before the parameter's description. All The variables included in this list can be modified by means of IDL instructions.

The input parameter listing is divided in sections accordingly with the different kind of variables that control the computational and physical environment of the simulations. These sections are

Run control. Includes all the parameters controlling the conditions of the simulations, namely, the total number of showers, the number of showers per run, the number of runs per process and the (maximum) CPU time per run. The directives that control these variables are *dynamic,* and may therefore vary during the simulations. The quantities displayed in the input parameter listing correspond thus to instantaneous values of the mentioned parameters.

File names. A listing with the names of all the files that will be created during the simulations (excluding, of course, internal scratch files). A detailed description of the output files that can be created by the simulation programs, together with guidelines on how to manage them can be found in chapter 4 (page 69); we just give here a brief description of them:

Log file (***taskname*.lgf**). This file contains information about the events that took place during the simulations. It contains also a summary of the input parameters that were in effect. Most of the data that goes into the log file is also written into the standard output channel.

Summary file (***taskname*.sry**, also ***taskname*.tex**). Output summary. This includes general simulation data and all the tables that were printed using IDL directive **PrintTables**.

Exported data files (***taskname*.t*nnnn***). Plain text files containing output tables.

Task summary script file (***taskname*.tss**). File containing a summary of input and output data, written in a format suitable for processing with other programs.

Binary dump file (***taskname*.idf**). This file contains (in machine-dependent binary format) all the relevant simulation data. This file is periodically updated during the task processing. In the case of an interruption, it is possible to restart the simulations from the last update. The file is also useful to obtain relevant data after the simulation is completed, or even during it. This can be done with the help of the summary program **AiresSry**.

ASCII dump file (***taskname*.adf**). Portable version of the IDF file, written at the end of the task. Like the IDF file, this file can be processed with the summary program **AiresSry**.

Compressed output files (***taskname*.grdpcles** and/or ***taskname*.lgtpcles**). These files contain detailed particle data. The ground particle file, for example, consists of a series of records of all the particles that reached ground in specified circumstances. Thanks to the compressed data formatting used, it is possible to save a large number of particle records

```
>>>>
>>>> This is AIRES version V.V.V (dd/Mmm/yyyy)
>>>> (Compiled by . . . )
>>>> USER: xxxxx, HOST: xxxxx, DATE: dd/Mmm/yyyy
>>>>

> dd/Mmm/yyyy hh:mm:ss. Reading data from standard input unit
> dd/Mmm/yyyy hh:mm:ss. Displaying a summary of the input directives:

>>>>
>>>>          REMARKS.
>>>>

        JUST AN EXAMPLE

>>>>
>>>>          PARAMETERS AND OPTIONS IN EFFECT.
>>>>
>>>> "(D)" indicates that the corresponding default value is being used.
>>>>

                          Task Name: mytask

     RUN CONTROL:
             Total number of showers:          2
     (D)              Showers per run:   Infinite
     (D)            Runs per process:    Infinite
     (D)            CPU time per run:    Infinite

     FILE NAMES:
                           Log file: mytask.lgf
                   Binary dump file: mytask.idf
                    ASCII dump file: mytask.adf
              Compressed data files: mytask.grdpcles
                                     mytask.lgtpcles
               Table export file(s): mytask.tNNNN
                Output summary file: mytask.sry

     BASIC PARAMETERS:
     (D)                        Site: Site00
                                      (Lat:    .00 deg. Long:     .00 deg.)
     (D)                        Date: dd/Mmm/yyyy

                    Primary particle: Proton
                      Primary energy: 1.5000 PeV
                Primary zenith angle:    15.00 deg
     (D)         Primary azimuth angle:      .00 deg
     (D)        Zero azimuth direction: Local magnetic north
                     Thinning energy: 1.0000E-04 Relative
     (D)            Injection altitude: 100.00 km (1.2829219E-03 g/cm2)
                     Ground altitude: 297.96 m  (1000.000 g/cm2)
           First obs. level altitude: 16.383 km (100.0000 g/cm2)
            Last obs. level altitude: 1.1733 km (900.0000 g/cm2)
          Obs. levels and depth step:         41    20.000 g/cm2
```

Figure 3.2. *Sample AIRES terminal output.*

```
     (D)           Geomagnetic field: Off
     (D)         Table energy limits: 10.000 MeV to 1.1250 PeV
     (D)         Table radial limits: 50.000 m  to 2.0000 km
     (D)   Output file radial limits: 100.00 m  to 12.000 km (grdpcles)
     (D)                              100.00 m  to 12.000 km (lgtpcles)

     ADDITIONAL PARAMETERS:
     (D)      Individual shower data: Brief
               Cut energy for gammas: 200.00 KeV
                Cut energy for e+ e-: 200.00 KeV
              Cut energy for mu+ mu-: 1.0000 MeV
               Cut energy for mesons: 1.5000 MeV
             Cut energy for nucleons: 150.00 MeV
     (D)      Bartol inelastic mfp's: On
     (D)        Gamma rough egy. cut: 2.0000 MeV
     (D)         e+e- rough egy. cut: 2.0000 MeV
     (D)    Hadronic Mean Free Paths: SIBYLL
     (D)               SIBYLL switch: On

     MISCELLANEOUS:
     (D)    Seed of random generator: Automatic
     (D)           Atmospheric model: Linsley's standard atmosphere

>>>>
> dd/Mmm/yyyy hh:mm:ss. Beginning new task.
> dd/Mmm/yyyy hh:mm:ss. Initializing SIBYLL 1.6 package.
  Initialization of the SIBYLL event  generator

  . . . (eventual output from SIBYLL) . . .

> dd/Mmm/yyyy hh:mm:ss. Initialization complete.
> dd/Mmm/yyyy hh:mm:ss. Starting simulation of first shower.
> dd/Mmm/yyyy hh:mm:ss. End of run number 1.
  CPU time for this run: . . . .
> dd/Mmm/yyyy hh:mm:ss. Writing ASCII dump file.
> dd/Mmm/yyyy hh:mm:ss. Task completed.
  Total number of showers: 2
> dd/Mmm/yyyy hh:mm:ss. Writing summary file.
> dd/Mmm/yyyy hh:mm:ss. End of processing.
```

Figure 3.2. *(continued)*

using a moderate amount of disk space. The format is universal, so the files can be written by a given machine and processed in a different one. The AIRES system includes a library of subroutines to process such files (see section 4.2).

Basic parameters. A list of geometrical and physical shower parameters. These variables define the initial conditions of the shower simulations (primary particle, axis inclination, etc.), as well as the settings that are in effect for the parameters of the monitoring algorithms (number of observing levels, range of radial distances for output files, etc.).

Additional parameters. Other shower parameters, generally depending on the model used. Since the interactions models are replaceable, the type and number of additional parameters may vary when changing simulation programs. The variables included in this section as well as the directives that allow controlling them may also be changed in future versions of AIRES. By default, only the most relevant parameters[4] are listed: Quantities associated with *hidden* IDL directives (see appendix B) are not included. Nevertheless, AIRES can be instructed to produce a full listing, by means of the directive: **InputListing Full**.

Miscellaneous parameters. Other parameters not included in the preceding sections.

3.3 More on IDL directives

3.3.1 Run control

In the example of section 3.2.5 (page 39), no specifications are made about the duration of processes and runs. This fact shows up in the variables listed in the run control section of the listing of figure 3.2 (page 46), when the default setting, "Infinite" is in effect for the number of showers per run, the number of runs per process and the CPU time per run. With such settings, the auto-save mechanism for fault tolerant processing is disabled: The IDF file will be saved only after finishing all the simulations specified with the input directives.

This can be acceptable for a short simulation in a reliable computer system. For heavy tasks it is recommended to split the simulations into processes and runs. It is worthwhile mentioning that *the auto-save/restore operations **do not** alter the results of the simulations,* which are bitwise identical independently of the number of such operations performed.

The IDL directives **ShowersPerRun**, **MaxCpuTimePerRun** and **RunsPerProcess** provide effective control on the computational conditions of the simulations. The following examples illustrate how them can be used.

```
RunsPerProcess 1
MaxCpuTimePerRun 2 hr
```

These two instructions indicate that a new run should begin every two CPU hours. Since the number of runs per process is 1, a new run will also imply the beginning of a new process; in other words,

[4] This also includes all the variables that were explicitly set by means of the corresponding IDL instructions.

the input file will be scanned every two CPU hours, allowing for eventual changes in the dynamical parameters of the simulations.

```
RunsPerProcess 4
ShowersPerRun 5
```

Here the maximum CPU time is not set, indicating that there will be no time limit for a run to complete. Instead, every run will finish after concluding the simulations of five showers. The processes will end when four runs are completed.

The three directives can also be used simultaneously:

```
RunsPerProcess 2
ShowersPerRun 2
MaxCpuTimePerRun 6 hr
```

These instructions indicate that a run will finish after six processing hours or after completing two showers, *what happens first.*

The run control directives –like any other *dynamic* directive– can be modified during the simulations if needed. The changes will be effective after a new process is started (see section 3.1). Let us assume that a certain task is started with the control parameters of the previous example. After a while it is decided that the maximum cpu time per run is too high and that there is no need to limit the number of showers per run. The input file is thus modified: (i) The **MaxCpuTimePerRun** line is replaced by

```
MaxCpuTimePerRun 3 hr
```

(ii) The **ShowersPerRun** line is deleted. After finishing the current process (with the old settings this may demand up to 12 CPU hours), and restarting the simulation program, the input file is scanned again and the new settings will become effective. The changes experimented by these dynamic parameters will be recorded in the log file (extension **.lgf**) in the following way

```
. . .
> dd/Mmm/yyyy hh:mm:ss. Reading data from standard input unit
> dd/Mmm/yyyy hh:mm:ss. Changing maximum number of showers per run.
  From: 2 to: Infinite
> dd/Mmm/yyyy hh:mm:ss. Changing maximum cpu time per run.
  From: 6 hr to: 3 hr
. . .
```

The dynamic directives can be changed as many times as needed, including the total number of showers (controlled by directive **TotalShowers**) which can be modified either during the simulations or after completing them to append new showers to an already finished task[5].

It is important to remark that the mechanism of dividing a task in several process is possible because all the relevant simulation data is saved into the internal dump file, and recovered in successive invocations of AIRES.

[5]Notice however that it is not possible to append new showers to any task that was initialized with a *previous* version of AIRES.

In some applications, however, it is necessary to completely disable this mechanism, and force AIRES to start a new task every time a new new process starts. This can be done with the help of the **ForceInit** directive, like in the following example:

```
Task myname
ForceInit
. . .
```

On the first invocation of AIRES, the task **myname** will be initialized and executed accordingly with the input directives. In a second call, the AIRES initializing procedures will check for the existence of the file **myname.idf**. After finding it, the task version will be increased by one, producing a IDF named **myname_001.idf**, and then the new task will be executed normally. On successive calls, the version number will be increased repeatedly, until finding the first non-existent file with name **myname_vvv.idf**.

3.3.2 File directories used by AIRES

The simulation programs read and/or write several files that contain different kinds of data. By default, all the files generated by AIRES are located in the ***working directory,*** defined as the current directory at the moment of invoking AIRES.

There are certain cases, however, where this setting is not adequate. For that reason, the IDL instruction set contains directives allowing to control the placement of AIRES files.

Let us first define the set of directories used by the AIRES system during the simulations:

Global. Containing the log, IDF, ADF and summary files.

Compressed output. Sometimes referred simply as ***Output directory,*** contains the compressed output files.

Export. Containing all the exported data files.

Scratch. Containing most of the internal files that are generated during the simulations, including the particle stack scratch files.

The output and scratch directories default to the current working directory when not specified. On the other hand, the global and export directory default to the current setting of the output directory.

The IDL directive **FileDirectory** permits complete control on the listed directories. For example, the sequence of instructions:

```
FileDirectory Scratch /mytmpdir
FileDirectory Export /myexportdir
```

sets the scratch (export) to the strings (must be meaningful to the operating system) **/mytmpdir** (**/myexportdir**). The directory specifications may be either absolute or relative. Relative specifications are always with respect to the working directory. In the preceding example the remaining directories are not specified, and will therefore take their respective default settings.

The directive

```
FileDirectory All /mydir
```

simultaneously sets the global, output and export directories.

There is an additional set of directories that can be specified while scanning the input data. The following instructions, for instance,

```
InputPath Insert /dir1:/dir2
InputPath Append /dir3
Input myinputfile.inp
```

will cause AIRES to search for file **myinputfile.inp** in the current working directory *and* –if not found there– in all the directories specified by means of the **InputPath** directives (notice the two alternative syntaxes). The search path is initially set to a string that contains the **bin** and **airesinputs** subdirectories included within the AIRES distribution. To eliminate those directories from the file search path (in case you really need it!) you can invoke the **InputPath** directive with no arguments to clear the search path, or without the **Insert** or **Append** modifiers to set the path to *exactly* the specified directories as in the following example:

```
InputPath /dir1:/dir2 ...
```

In this last case, input files will be searched for within the current directory, and the directories specified by the **InputPath** directive.

3.3.3 Defining the initial conditions

There are two mandatory specifications related to shower parameters that must always appear within the input data, namely, primary particle kind and energy.[6]

These two specifications, together with other related ones permit a very wide range of specifications for the shower parameters. Let us investigate some of the possible alternatives.

Mixed composition

The primary particle needs not be unique. AIRES allows for simulating showers with different primary particles each. The following example illustrates this feature:

```
PrimaryParticle Proton 0.6
PrimaryParticle Iron 0.4
```

With such settings, the primary will be proton (iron) with 60% (40%) probability. This means that in 100 simulated showers, approximately 60 will be proton showers while the remaining ones will have iron primaries.

[6] When using special primary modules the primary energy can be defined dynamically from the external module every time it is invoked. In this case there in no obligation of specifying it via a IDL directive.

If n alternative primary particles, p_i, $i = 1, \ldots, n$ were defined, with weights w_i ($w_i \neq 0$), then the probability for any shower of being initiated by particle p_j, $1 \leq j \leq n$ is given by

$$P_j = \frac{|w_j|}{\sum_{i=1}^{n} |w_j|} \tag{3.1}$$

Therefore, the weights entered in the IDL directives need not be normalized.

Besides this mixed composition feature, AIRES allows also to define *special* primary particles processed by external modules. For details see section 3.5 (page 63).

Varying energy

The directive

PrimaryEnergy $E_{\min}$ $E_{\max}$ γ

(see page 123), indicates that the primary energies will be in the interval $[E_{\min}, E_{\max}]$, selected with probability [28]:

$$p(E)\, dE = U^{-1}\, E^{-(\gamma+1)}\, dE, \quad E_{\min} \leq E \leq E_{\max}, \tag{3.2}$$

where

$$U = \int_{E_{\min}}^{E_{\max}} E^{-(\gamma+1)}\, dE = \begin{cases} \frac{1}{\gamma}\left(E_{\min}^{-\gamma} - E_{\max}^{-\gamma}\right) & \gamma \neq 0 \\ \\ \ln\left(E_{\max}/E_{\min}\right) & \gamma = 0 \end{cases} \tag{3.3}$$

γ can take any value. If not specified it is taken as 1.7.

Zenith and azimuth angles

The zenith angle directive placed in the example of figure 3.1 (page 41) corresponds to setting the angle to a fixed value. In this case the azimuth angle defaults to zero. On the other hand, the instruction

PrimaryZenAngle `0 deg 72 deg S`

indicates that the zenith angle distributes from 0° to 72° with *sine* distribution[7]

$$P_{\text{sine}}(\Theta)\, d\Theta = U^{-1}\, \sin\Theta\, d\Theta, \quad \Theta_{\min} \leq \Theta \leq \Theta_{\max}, \tag{3.4}$$

where

$$U = \int_{\Theta_{\min}}^{\Theta_{\max}} \sin\Theta\, d\Theta = \cos\Theta_{\min} - \cos\Theta_{\max}. \tag{3.5}$$

An alternative to the **S** specification is the **SC** (or **CS**) specification which corresponds to a *sine-cosine* distribution:

$$P_{\sin\cos}(\Theta)\, d\Theta = U^{-1}\, \sin\Theta\cos\Theta\, d\Theta, \quad \Theta_{\min} \leq \Theta \leq \Theta_{\max}, \tag{3.6}$$

[7]The sine distribution is sometimes called cosine distribution, relating it with the accumulative probability function of the sine distribution: $F_{\text{sine}}(\Theta) = \int_0^{\Theta} P_{\text{sine}}(u)\, du$.

where

$$U = \frac{1}{2}\int_{\Theta_{\min}}^{\Theta_{\max}} \sin(2\Theta)\, d\Theta = \frac{1}{4}\left[\cos(2\Theta_{\min}) - \cos(2\Theta_{\max})\right]. \tag{3.7}$$

For varying zenith angles, the default for the azimuth angle is to uniformly distribute in the interval $[0°, 360°]$. In this case the sine distribution corresponds to showers with directions having a uniform solid angle distribution.

The azimuth angle can also be set as a varying angle. The directive

```
PrimaryAzimAngle 37.2 deg 39.5 deg
```

indicates that the azimuth Φ will be uniformly distributed in the interval $[37.2°, 39.5°]$.

Using simultaneously instructions for both the zenith and azimuth angles, it is possible to simulate showers coming from a determined direction in the celestial sphere.

As pointed out in section 2.1.1 (page 9), the x-axis (zero azimuth axis) corresponds to the local *magnetic* north. If desired, it is possible to specify *geographic* azimuths:

```
PrimaryAzimAngle 37.2 deg 39.5 deg Geographic
```

In the preceding directive, the **Geographic** keyword indicates that the origin of the azimuth angles is the direction of the local geographic north. It is worthwhile mentioning that this *does not alter* the axis definitions of section 2.1.1; when geographic azimuths are in effect, the azimuth with respect to the AIRES coordinate system, Φ, is evaluated via

$$\Phi = \mathrm{D} - \Phi_{\mathrm{geographic}} \tag{3.8}$$

where D is the geomagnetic declination angle defined in section 2.1.5 (page 17). Notice that positive geographic azimuths indicate eastwards directions. For a complete description of this directive see page 123.

Position of injection, ground and observing levels

The directives **InjectionAltitude** (or its synonym **InjectionDepth**), **GroundAltitude** (or its synonym **GroundDepth**) and **ObservingLevels** permit controlling the position of the injection point, the ground surface and the different observing levels, respectively.

All the altitude specifications refer to *vertical altitudes,* noted as z_v in figure 2.1 (page 9), and can be expressed either as lengths (above sea level) or vertical atmospheric depths. Whenever necessary, AIRES transforms lengths into vertical depths and vice-versa using the current atmospheric model.

Notice that the vertical altitudes are equal to the corresponding z-coordinates only for points located in the z-axis. To illustrate this point, let us consider the following instructions

```
InjectionAltitude 100 km
GroundAltitude 1000 m
PrimaryZenAngle 60 deg
```

With such specifications, the primary particles will be injected at an altitude of 100 km above sea level, measured along the vertical passing by the injection point. Taking into account that the shower axis has an inclination of 60 degrees, and applying equation (2.1), it is possible to calculate the z-coordinate of the injection point, also referred as *central injection altitude.* In this case the result is $z_c = 17962$ m.

The positions of the observing levels defined in section 2.2.3 (page 23) can be set using **ObservingLevels**. This directive has two different formats:

(i) **ObservingLevels** N_o, with N_o an integer not less than 4.

In this case the positions of the observing levels are set taking into account the injection and ground vertical depths. Let X_i (X_g) be the injection (ground) depth, then the spacing between observables and the positions of the first and last observing levels are set via

$$\begin{aligned} \Delta X_o &= \frac{X_g - X_i}{N_o + 1} \\ X_o^{(1)} &= X_i + \Delta X_o \\ X_o^{(N_o)} &= X_g - \Delta X_o \end{aligned} \tag{3.9}$$

(ii) **ObservingLevels** N_o X_a X_b, with N_o an integer not less than 4 and X_a and X_b valid vertical depth or altitude specifications ($X_a \neq X_b$).

In this second case the positions of the first and last observing levels are set accordingly with X_a and X_b, with no dependence on the positions of the injection and ground levels:

$$X_o^{(1)} = \min(X_a, X_b), \quad X_o^{(N_o)} = \max(X_a, X_b). \tag{3.10}$$

The spacing between consecutive levels is evaluated using equation 2.19.

3.3.4 Atmosphere

In AIRES the profile of atmospheric depth and density is conveniently modeled via multilayer parameterizations like the Linsley described in section 2.1.2.

The main directive to set the atmospheric profile is **Atmosphere**. For example,

```
Atmosphere SouthPoleAvg
```

sets as current atmospheric model the predefined model identified by the string "**SouthPoleAvg**". The available predefined models are listed in page 109. The default atmospheric profile is **Linsley**, and corresponds to a parameterization of the US standard atmosphere [19].

Some of the predefined model accept parameters. Consider the following example:

```
Atmosphere Isothermic  Temp 28 C   Dens0 1.221 kg/m3
```

Directive	*Args*	*Action*
AddLayer	$h_{\rm beg}$ $h_{\rm end}$ $\rho_{\rm beg}$ $\rho_{\rm end}$	Adds an exponential layer.
AddLinLayer	$h_{\rm beg}$ $h_{\rm end}$ $\rho_{\rm beg}$ $\rho_{\rm lay}$	Adds a linear layer (for advanced users).
AtmDefault	Model id	Sets the atmospheric model to use to provide data corresponding to **MatchDefault** qualifiers or automatically added layers.
AtmIdent	Model id string	Sets the model identification string (maximum 16 characters).
AtmName	Model long name	Sets the model name (maximum 42 characters).
End		Ends processing of **AddAtmosModel** instructions.

Table 3.3. *Instructions recognized by the IDL directive* **AddAtmosModel**.

In this case the **Atmosphere** directive sets a isothermic model specifying a temperature of 28 C, and a density at sea level of 1.221 kg/m^3. The parameters may be specified in any order, and in case of missing specification a default value is provided.

Besides selecting among the predefined models, it is possible to add user-defined custom models by means of the **AddAtmosModel** directive. This directive allows to define a multilayer model by specifying the beginning and ending altitudes of each layer together with the densities at both layer ends. Then AIRES uses those data to calculate the multilayer coefficients, and include the defined model in the list of available ones. In case the user-specified layers do not cover the minimum range of altitudes going from sea level up to 100 km.a.s.l., AIRES will automatically add layers (using the default atmospheric model) to complete the description.

Consider the following example:

```
AddAtmosModel &mylabel
 AtmIdent  MyModelIdStr
 AtmName   My atmospheric model
 AtmDefault  Linsley
 AddLayer     0 m    800 m   MatchDefault     1.2184    kg/m3
 AddLayer   800 m   4000 m   1.2184 kg/m3     0.8422    kg/m3
 AddLayer  4000 m     12 km  0.8422 kg/m3     0.2765    kg/m3
 AddLayer    12 km    35 km  0.2765 kg/m3     6.4846E-6 g/cm3
 AddLayer    35 km   100 km  6.4846E-6 g/cm3  MatchDefault
&mylabel
```

In this case the input data for the **AddAtmosModel** directive is placed just after the directive as a *here-document* delimited by the label **&mylabel**. The format of the instructions is self-explaining. Each **AddLayer** instruction requires four quantities, namely, layer beginning and ending altitudes,

and the corresponding densities. The information that the user must provide with these instructions is redundant, since in the intermediate boundaries a same value must be specified twice. The advantages of more robustness and human readability amply compensate the extra effort of duplicating some data. Additionally, the **AddLayer** instructions can be placed in any order. Notice also that both the altitudes and densities are specified with two fields in all cases (number + unit). Any of the length or density units recognized by AIRES can be used (see table 3.1).

Once the model is defined via the **AddAtmosModel** directive, it can be set as the current model with the **Atmosphere** command

```
Atmosphere MyModelIdStr
```

Table 3.3 contains a listing of all the instructions recognized by directive **AddAtmosModel**. The demonstration examples that are included in the AIRES distribution contain several additional examples that explain in more detail how to define a custom atmospheric model.

3.3.5 Geomagnetic field

The components of the Earth's magnetic field used by the simulation programs can either be set manually or calculated with the help of the IGRF model [15] (see section 2.1.5). With the help of this model it is possible to obtain an accurate estimation of the geomagnetic field in a given geographic location and for a determined date.

To activate this mechanism for "automatic" evaluation of the magnetic field, it is necessary to specify both a geographic place and a date.

The directive **Site** tells AIRES the name of the site selected for the simulations. For example,

```
Site SouthPole
```

indicates that the selected place is "**SouthPole**". This name is one of the predefined locations that form the *AIRES site library*. Besides "SouthPole", this library initially contains several other sites related with air shower experiments. All the predefined sites are listed in table 3.4.

To specify a site that is not included among the predefined ones, it is first necessary to append it to the site library by means of the **AddSite** directive. Let us consider, for instance, the following directive:

```
AddSite cld -31.5 deg -64.2 deg 387 m
```

A new site "**cld**" is defined. The command parameters represent, respectively, the latitude, longitude and altitude above sea level that correspond to the defined site. The name string cannot contain more than 16 characters; names are case sensitive and must be different to all the previously defined ones.

The **Date** directive defines the date of an event. There are two alternative syntaxes, as displayed in the following examples:

```
Date 2018.2
Date 2018 3 1
```

Site name	***Latitude***	***Longitude***	***Altitude*** (m.a.s.l)
Site00	0.00°	0.00°	0
SouthPole	90.00° S	0.00°	2835
Malargue	35.20° S	69.20° W	1425
ElNihuil	35.20° S	69.20° W	1400
TelescArray	39.30° N	112.91° W	1400
AGASA	35.78° N	138.50° E	900
CASKADE	49.09° N	8.88° E	112
Dugway	40.00° N	113.00° W	1550
ElBarreal	31.50° N	107.00° W	1200
FlysEye	41.00° N	112.00° W	850
HaverahPark	53.97° N	1.64° W	220
Puebla	19.50° N	98.00° W	2200
SydneyArray	30.50° S	149.60° W	250
Yakutsk	61.70° N	129.40° E	850

Table 3.4. *Predefined sites of the AIRES site library. Site names are case sensitive. The data for Haverah Park, Sydney Array and Yakutsk sites come from reference [34].*

In the first statement the date is given as a floating point number taking the year as the time unit, while in the second the format "year month day" is used.

There are no special restrictions on the date specification. However, the IGRF database implemented in the current AIRES version contains data for the years 1955 to 2020. For dates outside that interval it is necessary to extrapolate the corresponding data in order to evaluate the geomagnetic field. This may lead to inaccurate estimations for dates very far from the validity range of the model (more than ten years away). Nevertheless, extrapolations near the given boundaries are acceptable, and are of course necessary for calculations beyond the year 2015.[8]

In case of missing date specification, it is set accordingly with the system time at the moment of starting the simulations.

Once a site and a date are set, the Earth's magnetic field will be calculated by means of the IGRF model, unless it is explicitly set by means of the **GeomagneticField** directive. Let us analyze some examples (see also page 116):

```
GeomagneticField Off
```

With this instruction the effect of the magnetic field on the motion of the charged particles will not be taken into account. However, the field will still be evaluated in order to determine the declination angle, which is used to transform geographical azimuths into magnetic ones (see page 53).

```
GeomagneticField 32 uT -60 deg 2 deg
```

[8]The next generation of IGRF data will be released after the year 2020.

The preceding directive instructs AIRES to fully override the IGRF estimation with the values indicated in the parameters, which respectively correspond to F, I and D (see section 2.1.5). Partial overriding is also supported, like in the following instruction

```
GeomagneticField 32 uT
```

The field strength, F, will be set to the value indicated in the first parameter, while I and D will remain as given by the IGRF model.

xz-plane Gaussian fluctuations, either absolute or relative, are also supported:

```
GeomagneticField 32 uT Fluctuation 500 nT
GeomagneticField On Fluctuation 10 %
```

Notice that fluctuations can be introduced with or without overriding the IGRF field components. It is also possible to specify **0.1 Relative** instead of **10 %**.

When magnetic fluctuations are in effect, then the magnetic field used for each shower will be different. Let $\mathbf{B}_0$ be the "central" value coming from the IGRF model and/or entered manually. Let ΔB be the specified fluctuations. Notice that in the case of relative fluctuations, ΔB is set using the field strength B_0: $\Delta B = B_0 \Delta B_{\rm rel}$.

Then for each new shower, two independent, Gaussian-distributed random numbers, ΔB_x and ΔB_z, having mean zero and standard deviation $\Delta B/\sqrt{2}$, are generated; and the magnetic field components are set via

$$\begin{aligned} B_x &= B_{0x} + \Delta B_x, \\ B_z &= B_{0z} + \Delta B_z. \end{aligned} \tag{3.11}$$

Notice, however, that the declination angle used for azimuth transformations will always come from the central value, that is, is not affected by the fluctuations introduced.

3.3.6 Statistical sampling control

The thinning algorithm described in section 2.3 (page 25) makes use of several external parameters that can be set by means of IDL directives. The thinning energy $E_{\rm th}$ is the most important parameter of the thinning algorithm. As illustrated in figure 3.1 (page 41), the directive **ThinningEnergy** permits setting $E_{\rm th}$, either absolutely or relative to the primary energy.

The directive **ThinningWFactor** allows controlling the maximum weight parameter $W_{\rm max}^{(EM)}$ defined in section 2.3 (page 25). The specification

```
ThinningWFactor 2.5
```

sets the *weight factor,* W_f, of equation (2.23) to 2.5, to be used with electromagnetic particles.

Recommended values for W_f are in the range 0.1 to 50; the default value is 12. Setting $W_f > 100$ is practically equivalent to $W_f \to \infty$ (see section 2.3.3).[9]

[9] **IMPORTANT:** The statistical weight factor of the AIRES extended thinning algorithm is not equivalent to the parameter with the same name defined for AIRES 1.4.2 or earlier. Therefore, the recommended values placed in the AIRES 1.4.2 manual [18] do not apply for the current version.

The weight factor that is used with non electromagnetic particles, $W_f^{(H)}$, can also be set by the user: The directive **EMtoHadronWFRatio** permits setting the ratio A_{EH} defined in equation (2.24). The default value $A_{EH} = 88$ is normally adequate, but some applications may require performing simulations with a different relation between electromagnetic and non electromagnetic weight factors, and in such cases the mentioned directive is useful to change the ratio as needed.

3.3.7 Output table parameters

The output tables listed in appendix C (page 132) are automatically calculated during the simulations, and the directives to retrieve these data will be explained in chapter 4 (page 69). Many of these tables can be customized by means of IDL instructions.

The number of observing levels defined for the longitudinal tables (table numbers 1000 to 1999) can be controlled using the IDL directive **ObservingLevels**, as already explained in section 3.2.5 (page 39).

The lateral distribution tables (table numbers 2000 to 2499), the energy distribution tables (table numbers 2500 to 2999), and the mean arrival time distribution tables (table numbers 3000 to 3499) are defined, by default, as histograms with 40 logarithmic bins (either radial or energy bins depending on the distribution type), plus two additional "underflow" and "overflow" bins.

The IDL directives **RLimsTables** and **ELimsTables** allow to control the radial and energy bins, respectively, as illustrated in the following examples:

```
RLimsTables 20 m 2 km
ELimsTables 2 MeV 1 TeV
```

The first directive sets the range for the standard lateral distributions. The lowest end of bin 1 (highest end of bin 40) is set to 20 m (2 km). The "underflow" bin will thus correspond to all entries with distances less than 20 m, while the "overflow" one to all entries beyond 2 km.

In a completely similar way, the second directive sets the lower and upper bounds for the 40 bin energy distributions, and the respective "underflow" and "overflow" bins.

With the current version of AIRES it is possible to save the tables in a shower per shower basis, besides the traditional average tables that have been always available. Since this may generate large IDF or ADF files in certain cases, the mechanism of individual shower table saving is disabled by default. The directive

```
PerShowerData Full
```

must be used to ensure that the individual shower tables are being saved.

3.3.8 Random number generator

The AIRES random number generator must be initialized before starting any set of simulations. The default action is to use a internally generated seed, generated with an elementary random number generator that uses the current clock and CPU usage registers. Therefore, different invocations of

AIRES with the same input directives, will generally originate different output data because of different initializations of the random number generator.

The default behavior can be changed if needed. The directive **RandomSeed** allows the user to set the random seed to a given number, or to get the seed from an already initialized task. These features are illustrated in the following examples:

1. The directive

   ```
   RandomSeed 0.1298004637
   ```

   sets the random seed to a fixed constant. The number must be greater than zero and less than one.

2. The directive

   ```
   RandomSeed GetFrom otheridfile
   ```

   extracts the seed used in the task that created the IDF file **otheridfile**, and uses it to initialize the generator.

3.4 Input parameters for the interaction models

The expression *interaction models* identifies a series of subroutines and functions that contain the actual implementations of the algorithms that control the propagation of particles. Such algorithms emulate the physical rules associated with the different interactions that take place in an air shower.

As it is well-known, there are still many open problems in this area and therefore the interaction models cannot be considered a crystallized part of the simulation programs. Furthermore, in the design of the interaction models and external packages units shown in figure 1.1 (page 7), every effort was made to make them easily replaceable, in order to be able to incorporate improved code to be developed in the future.

The IDL directives that are going to be mentioned in this section allow the user to control different model parameters. Such directives are defined from within the interaction model section, and for the reasons explained in the preceding paragraph, they are of a changing nature: For AIRES versions later than the current version 19.04.08 the model related directives may no longer be supported, be replaced by alternative ones or their syntax be totally or partially changed.

3.4.1 External packages

The last versions of EPOS [6], QGSJET [7], and SIBYLL [11] hadronic collisions packages are implemented in AIRES. For technical reasons they are compile-time implemented, and are available by means of different executable programs: **AiresS23** is a executable simulation program linked to SIBYLL 2.3, etc.

The current version of AIRES (19.04.08) includes links to EPOS LHC and 1.99, QGSJET-II 04 and 03, and SIBYLL 2.3, 2.3c, and 2.1.

All the particle-nucleus and nucleus-nucleus interactions with projectile kinetic energy above a certain threshold are processed using the external package, while the low energy ones are calculated by means of the extended Hillas splitting algorithm [4, 28], or a built-in nuclear fragmentation model, in the cases of hadron-nucleus or nucleus-nucleus collisions.

The IDL directive **ExtCollModel** is an On-Off switch that allows controlling the use of the external package (EPOS, QGSJET, or SIBYLL, depending on the executable program being used). The minimum energy required for the external package to be invoked can be altered using directives **MinExtCollEnergy** and/or **MinExtNucCollEnergy**, as in the following example:

```
MinExtCollEnergy 300 GeV
MinExtNucCollEnergy 500 GeV
```

AIRES supports also the directive **ForceModelName** that is useful to ensure that a given input data set will be processed only with a determined simulation program. For instance, if an input data set containing the instruction

```
ForceModelName QGSJET-II-04
```

is processed with other simulation program different from **AiresQIIr04**, the process will immediately be aborted with an error message. When the directive is not used no check is performed and the simulations can be started with any program.

The cross sections used to determine the collision mean free paths can also be controlled. In the current version there are several sets of hadronic cross sections available, namely, Standard, EPOS, QGSJET and SIBYLL (all versions) cross sections. The

The default mean free paths are the ones corresponding to the external hadronic package linked to the simulation program. The following example illustrates how to alter the default settings:

```
MFPHadronic SIBYLL23c
MFPThreshold 120 GeV
```

These instructions imply that the "SIBYLL23c" mean free paths will be used for collisions with energies over 120 GeV, while the standard mean free paths will be used for the ones with lower energies.

The mean free path sets supported in AIRES 19.04.08are: **Standard**, **EPOS**, **EPOS-LHC3400**, **EPOS1990**, **QGSJET**, **QGSJET-II-04**, **QGSJET-II-03**, **SIBYLL**, **SIBYLL231**, **SIBYLL23c**, **SIBYLL21**. The generic names (**EPOS**, **QGSJET**, **SIBYLL**) refer to the sets associated with the newest installed version of the respective hadronic models.

The previous directives also indicate that the nucleus-nucleus mean free paths will be evaluated using special algorithms included within the external hadronic packages if the projectile's energy per nucleon falls above the specified threshold; otherwise the mean free path will be evaluated via a built-in procedure that calculates it by scaling adequately the proton-nucleus mean free path corresponding

to the model being used. The directive **ExtNucNucMFP** allows to disable the call to the external routine, and use the built-in algorithm for all projectile energies.

The hadron-nucleus/nucleus-nucleus and/or the photon-nucleus collisions can be disabled if desired:

```
NuclCollisions Off
PhotoNuclear Off
```

These settings are intended to be used only for special purposes: The results obtained in such conditions may be rather unphysical.

3.4.2 Other control parameters

There are several IDL instructions that allow controlling different parameters and/or processes of the simulation algorithms. These IDL directives need not be used for normal operation. Furthermore, the user should take into account that improper settings for some of the parameters associated with these instructions may lead to unphysical results.

PropagatePrimary. Logical switch to control the initial propagation of the primary.

SetTimeAtInjection. Logical switch to control whether or not the shower time is set to zero at the injection point. The shower clock can be set to zero at the injection point (default) or at the moment of the first primary interaction.

GammaRoughCut, ElectronRoughCut. Threshold energies for "normal" propagation of gammas and electrons, respectively. Particles with kinetic energies below those thresholds are "roughly" propagated, that is, many processes are calculated only approximately, or are ignored at all.

ForceLowEDecays, ForceLowEAnnihilation. These directives control the kind of action to be taken when low energy particles that can decay or undergo annihilation reach the low energy threshold.

LPMEffect. IDL switch to enable/disable the LPM [25, 30] effect. The default is **LPMEffect On**.

DielectricSuppression. IDL switch to enable/disable the dielectric suppression [25, 31] effect. The default is **DielectricSuppression On**.

MuonBremsstrahlung. IDL switch to enable/disable the muon bremsstrahlung and muonic pair production processes. The default is **MuonBremsstrahlung On**

AirZeff, AirAvgZ/A, AirRadLength. IDL directives associated with internal parameters. For a detailed explanation see appendix B (page 106).

Since most of these IDL instructions are *hidden* directives (see page 48) the respective settings in effect will not be included in the input data list, unless explicitly indicated by means of directive **InputListing** (see page 118). Additionally, warnings messages will be issued when using any directive which may lead to simulations with unphysical results.

3.5 Special primary particles

In many cases of interest, it is necessary to simulate showers that cannot be described adequately with the usual scheme of a single primary particle interacting with a nucleus in the atmosphere and generating a set of secondaries to be propagated. Instead, one has that a particular set of interactions that only affect the primary particle, originates a series of "normal" secondary particles that hit the atmosphere and originate the corresponding cascades. In general, such special interactions are not modeled adequately by AIRES propagating engine, but it is possible to overcome this difficulty allowing the simulation program to start a shower with *multiple* "primary" particles which are the secondaries coming out from the "special" interactions.

The following are examples where the mentioned scheme applies:

- An *exotic* cosmic particle (a cosmic neutrino, for instance) interacts and produces a series of particles that can be normally propagated by AIRES.
- An electromagnetic particle interacts with the Earth's magnetic field *before* reaching the atmosphere, and producing a *pre-shower* whose products finally reach the atmosphere and start interacting with it.
- A cosmic particle disintegrates (before reaching the Earth) in two or more fragments that arrive simultaneously in slightly distant points.
- Etc.

AIRES 19.04.08 allows the user to simulate showers initiated in such conditions. An external, user provided, program will be responsible for generating the particles to be injected at the beginning of the shower. This process is completely dynamic, and the sets of generated primary particles may vary from shower to shower.

To implement such an interface is very simple. The user needs to: (i) Define the special particle within the IDL instructions. (ii) Set up the external program that will be invoked (via a system call) at the moment of starting a new showers.

3.5.1 Defining special particles

The AIRES IDL directives allow to specify particles by names ("proton", "gamma", etc.). The set of known particle names can be expanded to include those special "particles" which need to be treated separately.

Consider the following examples:

```
AddSpecialParticle myparticX  Xpartsim
AddSpecialParticle myparticY  Xpartsim  type Y
```

The IDL directive **AddSpecialParticle** takes at least two arguments: (i) A *special particle name* that uniquely identifies the added special particle, and (ii) The name of the executable module that will be invoked when starting the showers initiated by the respective particles.

In the preceding example, two special particles, namely, **myparticX** and **myparticY** are defined and associated to the same external module, **Xpartsim**. In the case of the definition of **myparticY**, some arguments are specified (" **type Y**"). Such arguments are passed (portably) to the module.

Once the special particle(s) are defined[10], their names can be used as argument of the **Primary-Particle** directive:

```
AddSpecialParticle myparticX  Xpartsim
. . .
PrimaryEnergy 20 EeV
PrimaryParticle myparticX
```

Special particles can also be used in the case of mixed composition (see page 51), like in the following example:

```
AddSpecialParticle SSP1  module_1
AddSpecialParticle SSP2  module_2
. . .
PrimaryParticle SSP1 0.2
PrimaryParticle SSP2 0.3
PrimaryParticle Proton 0.5
```

In this case the primary will be **SSP1**, **SSP2**, or proton with probabilities 20%, 30%, and 50%, respectively.

3.5.2 The external executable modules

Every time a special primary shower is started, the simulation program will invoke the executable module associated with the corresponding primary, defined using the **AddSpecialParticle** directive. Such an executable program can be a FORTRAN, C or C++ program (or a shell script running it), and must be capable of providing the calling module with the list of primary particles that will be added to the particle stacks before starting the simulation of that shower.

The simulation program and the external module communicate via internal files in a way that is transparent for the user and completely portable.

The AIRES object library includes a series of user-friendly routines (callable from FORTRAN, C or C++) that ease the task of writing such external modules.

Figure 3.3 displays a brief FORTRAN program with the basic structure needed in every module capable of building a list of primary particles to start the simulation of a shower.

The program starts with a call to routine **speistart** and ends with a call to **speiend**. It is essential to maintain this structure in any external module: All the calls to any AIRES library routine *must* be placed within the mentioned calls.

Once the interface is started, the system is ready to accept primary particles that will be added to the primary particle list. The basic routine to add primaries to the list is **spaddp0**. For each invocation

[10] Up to ten different special particles can be defined for a given task.

```
c
c     An example of an external module to process "special" primary
c     particles.
c
      program  specialprim0
c
      implicit none
c
c     Declaration of variables retrieved when starting the interface
c     with the calling program.
c
      integer           shower_number
      double precision  primary_energy
      double precision  default_injection_position(3)
      double precision  injection_depth, ground_depth
      double precision  ground_altitude, d_ground_inj
      double precision  shower_axis(3)
c
      integer           rc
      double precision  urandomt
c
c     Some particle codes (AIRES coding system).
c
      integer           pipluscode, piminuscode
      parameter         (pipluscode = 11, piminuscode = -11)
c
c     FIRST EXECUTABLE STATEMENT.
c
c     Starting the AIRES-external module interface.
c
      call speistart(shower_number, primary_energy,
     +               default_injection_position, injection_depth,
     +               ground_altitude, ground_depth,
     +               d_ground_inj, shower_axis)
c
c     Injecting two particles at the initial injection point, and in
c     the direction of the shower axis.
c
      e1 = primary_energy * urandomt(0.05d0)
      e2 = primary_energy - e1
c
      call spaddp0(pipluscode, e1, 1, 0.d0, 0.d0, 1.d0, 1.d0, rc)
      call spaddp0(piminuscode, e2, 1, 0.d0, 0.d0, 1.d0, 1.d0, rc)
c
c     Completing the main program-external module interchange.
c     The integer argument of routine "speiend" is an integer return
c     code passed to the calling program. 0 means normal return.
c
      call speiend(0)
c
      end
```

Figure 3.3. *A sample module for processing special primary particles. The purpose of this example is to illustrate the basic structure of a program to process the special primaries; the programmed algorithm is not intended to have any validity from the physical point of view.*

of this routine, the corresponding particle is added to the internal list of particles. There is no limit in the number of primary particles that can be included in the mentioned list, but the sum of their energies must not be larger than the primary energy specified in the input instructions and stored in the variable **`primary_energy`** appearing in figure 3.3.

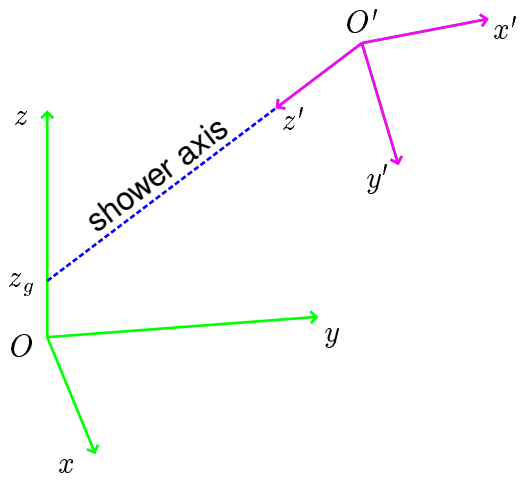


Figure 3.4. *The shower axis-injection point coordinate system, $x'y'z'$ (magenta), contrasted with the AIRES coordinate system, xyz (green). The origin of the AIRES coordinate system, O, is located at sea level, while O' is located at the original shower injection point. z_g is the ground altitude. The z'-axis is parallel to the shower axis, the x'-axis is always horizontal, and the $y'z'$-plane contains the z-axis.*

Arguments number 3 to 6 of routine **spaddp0** define the direction of motion of the corresponding particle. Argument number 3 is an integer switch selecting the coordinate system to use and the remaining quantities give the components of a vector, not necessarily normalized, pointing in the direction of motion of the particle. There are two options for argument number 3 (variable **csys** in the description of page 201):

0 To select the AIRES coordinate system defined in section 2.1.1 (page 9).

1 To select the *shower axis-injection point* system. This is a special coordinate system whose z-axis is parallel to the shower axis and its origin is placed at the original injection point (which remains uniquely determined by the shower zenith and azimuth angles and by the injection and ground altitudes). In this coordinate system, illustrated in figure 3.4, the coordinates $(0, 0, z')$ and the vector $(0, 0, 1)$ indicate, respectively, the position and direction of motion of a particle that moves along the shower axis and towards the ground.

The process is completed with the call to **speiend**. This ensures that all the relevant variables are transmitted back to the main simulation program, which will recover the control after the external module ends. Both **speistart** and **speiend** must be called only once within the entire external module.

Notice also that one of the AIRES random number routines, namely, **urandomt** (see page 213), is used to evaluate the energy if the π mesons being included in the list of primary particles. The random number generator is *not* initialized. Instead, its current status is passed by the main simulation program to the external module, and read-in within **speistart**. As a consequence, the generated random numbers will be different in different invocations of the external module. Routine **speiend** writes back the final status of the random number generator, and it is recovered by the main simulation program, so the numbers used in one and other program are always independent. If the AIRES random number generator is not used within the external module, then there are no alteration in the series of random numbers used by the main simulation program.

An actual external module to process special primaries will surely be much more complex than the one of the preceding example. The user can provide special routines with the procedures needed for that purpose, and use routines from the AIRES object library as well. Many of the modules described in appendix D (page 140) can be used within special primary programs, in particular the ones directly related with the special particle interface system, which provide a set of tools covering the needs of the most common situations, namely:

Retrieval of environmental information. Routine **speistart** starts the AIRES-external module interface and retrieves some basic variables, namely, shower number, primary energy, original injection position (three coordinates), vertical atmospheric depth of the original injection point, ground level altitude and vertical atmospheric depth, distance between the original injection point and the ground level (measured along the shower axis), and unitary vector in the direction of the shower axis. Besides these variables, it is possible (optionally) to retrieve additional ones calling other routines included in the AIRES object library:

- **speigetpars** (page 205) returns the parameter string that can be (optionally) specified in the IDL instruction that defines the corresponding special particle (see directive **AddSpecialParticle**, page 108). The simulation program passes the argument string directly, without making any special processing on it.
- **speigetmodname** (page 204) returns the name of the executable module specified in the definition of the corresponding special particle.
- **sprimname** (page 211) returns the name of the special particle corresponding to the current invocation of the external module.
- **speitask** (page 209) returns the current task name.
- **spnshowers** (page 210) returns three integers that correspond, respectively, to the total number of showers assigned to the task, and the numbers of the first and last showers. These quantities are related to the specifications entered with the directives **TotalShowers**

and **FirstShowerNumber**. The variable **shower_number** set when calling **speistart** (see figure 3.3), will always be equal or larger (smaller) than the first (last) shower number.

Adding primary particles to the primary particle list. Routine **spaddp0** appends to the particle list the particle defined with the arguments used in the corresponding call, as illustrated in the example of figure 3.3. Additionally, there are two other related library routines available, namely, **spaddpn** (page 202) to append with a single call a set of various primary particles, and **spaddnull** (page 200) to include *null* (unphysical) particles. A null particle is not included in the simulations, but its energy is added to the global null particle energy counter. Nuclei can be normally appended to the particle list. Nuclear codes can be conveniently evaluated using routine **nuclcode** (page 189).

Changing the injection coordinates and time. After the initial call to **speistart**, the injection point is set to the original injection point defined by the global parameters entered within the input data (zenith and azimuth angles, etc.). The coordinates with respect to the AIRES coordinate system of the original injection point are returned by **speistart** (this corresponds, in the example of figure 3.3, to array **default_injection_position**). The injection coordinates and time can be changed at any moment using routine **spinjpoint** (page 207).

Setting the point of first interaction. When using normal primary particles, AIRES evaluates automatically the atmospheric depth where the first major interaction takes place. This is not possible in the case of a special particle when a series of primaries are injected before starting the simulations; and the default action will be to set the first interaction at the original injection point, regardless whether that points corresponds or not to an actual point of interaction. As an alternative to the default action, it is possible to set manually the coordinates of the point of first interaction using routine **sp1stint** (page 199). Of course, this affects only the statistical analysis of the first interaction depth, and has no effect on the propagation of the particles.

Version of external module. The user can assign a version number to the external module. This version number must be passed to the main program by means of routine **speimv** (page 206). The version number is stored with all the information associated with the current shower, in particular in the compressed output files. It is strongly recommended to assign version numbers to external modules that will be used in production simulations.

We recall here that all the calls to every one of the routines listed in the previous paragraphs *must* be placed *after* the call to **speistart** and *before* the call to **speiend**.

The IDL directive **SpecialParticLog** allows to print information about the primary particles that are injected after each invocation of the external module. Notice that the default is to print no information in the log (extension lgf) file.

Chapter 4

Managing AIRES output data

4.1 Using the summary program AiresSry

Every time a task is completed, the simulation programs invoke some output procedures that create a *summary file* displaying a series of results related with the already finished simulations; a *task summary script file* can also be created. As it will be discussed in this section, there are several IDL directives that allow controlling such AIRES output data.

The summary program, **AiresSry**, which is part of the AIRES system allows the user to process the simulation data contained within the *internal dump file (IDF)* or, equivalently, the *portable dump file (ADF),* and retrieve any of the available observables, similarly as the main simulation programs do. It is worthwhile mentioning that **AiresSry** can be used *before* as well as *after* the simulations are finished. In the first case it is possible to monitor the development of the simulation task while the former alternative is most convenient for analysis tasks. Backwards compatibility is always ensured: Old IDF's or ADF's generated with any previous version of AIRES can be processed normally using **AiresSry**.[1]

Many observables are of "tabular" nature, that is, an array of data whose elements correspond to a set of values of a determined variable. For example, the longitudinal development of the number of gamma rays is represented by an array whose elements give the number of gamma rays that have crossed the different observing levels, as a function of the observing level altitude.

Most of the tabular observables commonly defined are automatically calculated by the simulation programs. The corresponding data arrays are stored in the IDF file and can be retrieved in several ways (see below). The complete list of currently available output data tables (more than 300) is placed in appendix C (page 132).

[1] Notice, however, that a set of *simulations* created using a determined version of AIRES must be ended using the same version.

4.1.1 The summary file

The summary file (extension **.sry**) can be divided into two parts: (i) The general section, which includes data on the evolution of the simulations as well as some basic shower observables. (ii) Tables section, containing data tables accordingly to user's specifications.

The summary file is generally written as a plain text file (this is the default). However, the IDL instruction **LaTeX** permits generating summaries that can be processed using the LaTeX typesetting system. If the LaTeX switch is enabled, then the AIRES system will generate two files, namely ***taskname*.sry** and ***taskname*.tex**. The last of these two files can be normally processed by a standard LaTeX system.

On the other hand, it is also possible to instruct any of the AIRES programs not to write the summary file. To do this, just include the directive **Summary Off** into the input data stream.

General section

The general section of the summary file begins with computer related information (task and user identification, CPU time usage, etc.). It also includes information about the input parameters used, and reports on the number of particle entries processed at each stack. A complete report on stack usage can be obtained using the IDL directive

```
StackInformation On
```

Then general information about the number and energy of particles reaching ground is displayed. For all the output observables, its mean[2], standard deviation, root mean square error of the mean, minimum and maximum, are reported.

The IDL directive **OutputListing Full** will generate an additional section containing information (generally of computational nature) on several output quantities defined for different algorithms.

The general section concludes with reports on the vertical depth of the first interaction, and on the location of the *shower maximum*.

The data collected for the longitudinal development of all charged particles, that is, the number of charged particles $N_c(i)$ that crossed the observing level i, for all $i = 1, \ldots, N_o$, is used to estimate the *shower maximum,* $X_{\max}$, here defined as the *vertical* depth of the point where the number of charged particles reaches its maximum. The number of charged particles at the maximum, $N_{\max}$ is also evaluated.

The estimation of the shower maximum is done by means of a **4-parameter** fit to the Gaisser-

[2]The statistical analysis is made in a shower-per-shower basis.

Hillas function [35][3]

$$N_{\rm ch}(X) = N_{\rm max}\left(\frac{X - X_0}{X_{\rm max} - X_0}\right)^{[(X_{\rm max}-X_0)/\lambda]} \exp\left(\frac{X_{\rm max} - X}{\lambda}\right), \quad X \geq X_0. \tag{4.1}$$

$X_{\rm max}$, $N_{\rm max}$, X_0 and λ are the free parameters to be adjusted. Notice that $N_{\rm ch}(X_0) = 0$,[4] and that $N_{\rm ch}(X)$ is taken as 0 for $X < X_0$.

A weighted nonlinear least squares fit performed with the aid of the very robust Levenberg-Mardquardt algorithm –as implemented in the public domain software library Netlib [16]– is done after the simulation of every individual shower is completed. The values reported in the summary file correspond to the plain average of all the fits with "reasonable" results (converged fits). The number of such converged fits is also reported.

Tables section

The output data tables listed in appendix C (page 132) that are automatically calculated during the simulations can be totally or partially included within the output summary file. An index of such tables can also be printed using the directive **TableIndex**

The **PrintTables** directive must be used to include one or more tables within the output summary file. Its syntax is shown in the following example:

```
PrintTables 1291 Options RM
```

This instruction orders AIRES to "print" table 1291 (longitudinal development of all charged particles) into the summary file. The options used are: **R**, to list RMS errors of the means, and **M** to include maximum and minimum data as numerical entries. For a detailed explanation of the directive **PrintTables** see page 124.

4.1.2 Exporting data

An interesting feature of both the summary and the main simulation programs, is that they are able to generate output files containing any one of the tables listed in appendix C (page 132). Let us consider,

[3]Our definition of the Gaisser-Hillas function involves *vertical* depths. Some authors, however, use *slant* depths instead. Both definitions can be used to parameterize the shower profile. Furthermore, notice that in the plane Earth approximation both "vertical" and "slant" forms are equivalent, provided the parameters are adequately interpreted, that is, taking into account the factor $\cos\Theta$ of equation (2.9). If the Earth's curvature is taken into account, the translations between vertical and slant quantities must be done numerically (see pages 14 and 214).

[4]The depth X_0 refers to the point where the Gaisser-Hillas function is zero, and is not equal and not even necessarily related to the *depth of the first interaction,* noted X_1 in this manual.

for instance, the following IDL instructions:

```
Task mytask
Summary Off
Export 1205 1211
Export 1293 Option a
Export 2001 Options ds
Export 2791 2793 Options ML
End
```

Here ***mytask*** is a string that represents an already finished or currently running task. The **Summary Off** directives disables the summary file. This is, of course, optional, but might be useful when the user is just interested in creating table files.

The first **ExportTables** directive (the abbreviated name will be correctly interpreted by any of the AIRES main programs) indicates that all the tables whose numbers are in the range $[1205, 1211]$ must be exported with the default options. Looking at the listing in appendix C (page 132), it comes out that the involved tables are tables number 1205, 1207 and 1211.

The second export directive instruct AIRES to export table 1211 with the option of listing the *slant* depths of the observing levels, that is, measured along the shower **a**xis (equation (2.8)). By default (Option **r**) all atmospheric depths listed within exported tables are vertical depths.

In the second export directive, the option string **ds** modifies the default settings. **d** indicates that the particle numbers must be normalized to particle densities, expressed in particles/m^2; and **s** suppresses the file header (only the tables will be written). This last option may be useful when the exported files are read by other applications (piped). Suppressing the file header, however, may lead to not understandable files, especially if they are not processed at the moment they are produced. It is therefore recommended to always keep such information; and it must be also taken into account that all the header lines are "commented out" by means of a leading comment character which defaults to "**#**", but can be changed by means of the directive **CommentCharacter** (see page 110).

In the last example, the energy distributions 2791, 2792 and 2793 are exported. The option **M** indicates that energies must be expressed in MeV (the default is GeV); while **L** indicates that the corresponding data are normalized to $dN/d\log_{10} E$ distributions. The alternative option **l** corresponds to $dN/d\ln E$ normalization.

To process the preceding code, it might be useful to edit a small text file containing them and then use –for instance– the summary program to process it:

```
AiresSry < myfile.inp
```

The files ***mytask*.t1205**, ***mytask*.t1207**, ..., etc., will be created. If such files already exist, they will be overwritten.

If the simulations that generated the data being processes were run with the **PerShowerData** option **Full** (see section 3.3.7), then it is possible to export *single shower tables* by placing the directive

```
ExportPerShower
```

together with the **ExportTables** one(s). Returning to our previous example, if such directive is placed inside the file **myfile.inp**, then for each one of the exported tables, the files

mytask*.t*nnnn
mytask*_s0001.t*nnnn
mytask*_s0002.t*nnnn
...

will be created, corresponding respectively to the usual average table, and the tables for shower 1, 2, etc.

4.1.3 The task summary script file

The task summary script file file (extension **.tss**) is a text file containing information about the main input and output parameters of the simulation, in a format suitable for further processing by other programs.

The format of this file is very simple: Each data item is written using a single line, in the format

`Keyword =` ***value***

Some comments are included to make the file more human readable. The comment lines begin with a comment character ('#' or the character set with the **CommentCharacter** directive).

In figure 4.1 a typical TSS file is displayed. Some of the records were not displayed for brevity. The first data items correspond to the version of AIRES used for the simulations, and the units corresponding to different quantities written in the file. The general data, basic and additional input parameters, and miscellaneous sections contain the current values of all the input parameters. The last section contains a summary of shower observables. Using one line per shower, a series of shower output data is displayed. The **ShowerPerShowerKey** line gives the key with the meaning of each one of the columns of the shower data lines, starting with the primary code (key **PrCode**) for the first item after the equal sign, and continuing until completing all the following items:

1	**PrCode**	Primary particle code
2	**PrEgy**	Primary energy
3	**Zenith**	Shower zenith angle
4	**Azim**	Shower azimuth angle
5	**X1v**	Vertical depth of first interaction
6	**Xmaxv**	Vertical depth of shower maximum
7	**Nmax**	Number of particles at shower maximum
8	**X0v**	Fitted parameter X_0 (vertical) of Gaisser-Hillas function (equation (4.1))
9	**Lambda**	Fitted parameter λ (vertical) of Gaisser-Hillas function (equation (4.1))
10	**SofSqr**	Normalized sum of squares (equation (D.1)) from the longitudinal profile fit
11	**FitRc**	Return code of the longitudinal profile fit

By default, AIRES does not create any task summary script file. The directive **TSSFile** must be used to enable this feature.

```
#### AIRES TSS --- version V.V.V
#>>>
#>>> This is AIRES version V.V.V (dd/Mmm/yyyy)
#>>> (Compiled by . . .)
#>>> USER: xxxxx, HOST: xxxxx, DATE: dd/Mmm/yyyy
#>>>
#
# TSS file for task mytask
#
#
# Program and compilation parameters.
#
AiresVersion = V.V.V
#
# Units
#
LengthUnit = m
TimeUnit = sec
EnergyUnit = GeV
DepthUnit = g/cm2
AngleUnit = deg
MagneticFieldUnit = nT
#
# General data
#
TaskName = mytask
TaskVersion = 0
#
TotalShowers = 3
CompletedShowers = 3
. . .
. . . (Other general data)
. . .
#
# Basic Input Parameters.
#
Site = Site00
SiteLatitude = 0.000000
SiteLongitude = 0.000000
EventDate = dd/Mmm/yyyy
#
NumberOfDifferentPrimaries = 1
PrimaryParticle = Proton
PrimaryParticleCode = 31
. . .
. . . (Other task input parameters)
. . .
#
# Parameters relative to each shower
#
ShowerPerShowerKey = PrCode PrEgy Zenith Azim . . .
#
DataSh000001 =     31   350000.       0.00000       0.00000 . . .
DataSh000002 =     31   350000.       0.00000       0.00000 . . .
DataSh000003 =     31   350000.       0.00000       0.00000 . . .
#
# End of tss file
#
```

Figure 4.1. *Sample AIRES task summary script (TSS) file.*

4.2 Processing compressed particle data files

Like other simulation systems [1, 36], AIRES can produce output files containing detailed information about the particles generated during the simulations. The well-known fact that that detailed information generates huge amounts of data has been especially taken into account in the design of AIRES, which includes an *ad hoc* data compressing algorithm to save file space.

A detailed explanation of the compressing algorithm –a rather technical matter– is beyond the scope of this manual. We shall limit ourselves to briefly list its main characteristics:

Format. The compressed files are plain text files that can be generated in any computer and copied and processed in any other one. This is valid even if the machines do not have the same operating system and/or do not use the same character codes (for example non-ASCII machines).

Organization. The files contain a header with data related to its structure and the conditions of the simulation. The particle data section represents the bulk of the file and, in general, the records corresponding to any one of the simulated showers are delimited by "beginning of shower" and "end of shower" records. There is practically no limit in the number of showers that can be included in a single file.[5] On the other hand, groups of showers can be saved into separate files, up to the limit of storing each shower in a different file (see page 127).

Compression rate. The data compression algorithms were designed to take profit of the physical properties of the quantities being stored. This involves information about lower and upper bounds for a variable, possibility of subtracting a given fixed value[6], etc. Precision requirements were also taken into account, imposing a minimum of five significant figures in most cases. To give an idea of the size of compressed records, let us consider the default ground particle record (see below) whose fields are: Particle code, logarithm of the energy, logarithm of the distance to the core, polar angle in the ground plane, arrival time, x and y components of the direction of motion, and statistical weight. This record thus has one integer field and six real ones, and its length is 18 characters (bytes). This figure should be contrasted with a standard FORTRAN internal write statement with single precision for real variables, which generates 28 data bytes when writing the same fields. Taking into account that such records usually include additional formatting fields, the compression rate of AIRES algorithm compared with standard unformatted FORTRAN i/o should be larger than 36%.

It is worthwhile mentioning the the AIRES package includes a library of subroutines, namely, the *AIRES object library,* which contains many routines to read and process the compressed output files. Backwards compatibility is always ensured: Old compressed files generated with any previous version of AIRES can be read normally using the library routines.

[5] It is possible to store up to 759375 showers in a single compressed file.
[6] This refers to internal operations which do not alter any user-level results.

	Field	*Name*	*Description*
Integer	1	Primary particle code	Stores the code of the primary particle.
	2	Shower number	Shower number. By default, the first shower is labeled with the number 1, but the user can manually set the first shower number by means of the IDL directive **FirstShowerNumber** (see page 114).
	3–8	Starting date and time	Six fields containing, respectively, the year, month, day, hours, minutes and seconds corresponding to the beginning of the simulation of the corresponding shower.
Real	1	Primary energy (GeV) (log)	The logarithm of the primary particle's energy.
	2	Primary zenith angle (deg)	The zenith angle of the primary particle.
	3	Primary azimuth angle (deg)	The azimuth angle of the primary particle.
	4	Thinning energy (GeV)	The absolute thinning energy used for the respective shower.
	5	First interaction depth (g/cm2)	The vertical depth of the point where the first interaction took place, X_1.
	6	Central injection altitude (m)	The z-coordinate of the primary's injection point (see figure 2.1).
	7	Global time shift (sec)	The time t_0 required for a particle moving along the shower axis at the speed of light, to go from the injection point to the ground level.

Table 4.1. *Fields contained in the "*beginning of shower*" record of compressed particle files. The structure of this record does not depend on the compile-time option selected for the particle record.*

4.2.1 Customizing the compressed files

Two kinds of compressed files are implemented in the current version of AIRES (19.04.08):

Ground particle file. (Extension **.grdpcles**) This file contains records with data of particles that reached the ground surface.

Longitudinal tracking particle file. (Extension **.lgtpcles**) Compressed file containing detailed data related with particles crossing the predefined observing levels (see section 2.2.3).

Ground particle file

There are three basic types of data records in this file: "Beginning of shower" record, "end of shower" record and particle record (also referred as default record). The "external primary particle" and "special primary trailer record" are also defined. These last two records are used only in connection with the special primary particles described in section 3.5 (page 63).

All the particle records written out during the simulation of a single shower will appear in the file preceded by a beginning of shower record, and followed by the corresponding "end of shower" one.

The fields that make the beginning (end) of shower record are listed in table 4.1 (4.2). Tables 4.3 and 4.4 describe the fields of the special primary related records. In these and in any other records, the data fields can be classified in *integer* and *real* fields.

The fields contained in such delimiting records account for general air shower parameters or observables and were included for special analysis tasks.

In the case of showers initiated by special primary particles (see section 3.5), the "Primary particle" code of the corresponding "beginning of shower" record will not correspond to a standard particle code. Instead, the returned code will be a negative integer with an absolute value slightly smaller than 100000.[7]

In those cases, the "beginning of shower" record will be followed by a series of "external primary particle" records (one for each injected primary particle). This series ends with a "special primary trailer record" which will precede the default particle records written for that shower.

The fields included in the default records, associated with particle data, can be selected at compile time among the various available alternatives. The installation configuration file (see appendix A) contains detailed instructions on how to select the particle record options.

The most relevant physical properties of the ground particles can be saved in the ground compressed file, namely,

Particle code. An integer code that identifies the particle.

Energy. The logarithm of the kinetic energy of the particle.

Coordinates. The polar coordinates (R, φ) of the particle at ground, measured from the intersection of the shower axis with the ground surface. R is the distance to the core and φ is the angle with respect to the x-axis.

Direction of motion. The x and y components, u_x, u_y, of the unitary vector $\mathbf{u}$ which indicates the particle's direction of motion. The u_z component must be negative for the particles reaching ground because such particles move downwards. It can be calculated via:

$$u_z = -\sqrt{1 - u_x^2 - u_y^2}. \tag{4.2}$$

[7] The library routine **crospcode** is the adequate one to manage such special particle codes.

	Field	***Name***	***Description***
Integer	1	Shower number	Shower number (matching the shower number of the corresponding "beginning of shower" record).
	2	Xmax fit return code	Integer code returned by the $X_{\max}$ and $N_{\max}$ fitting routine described in section 4.1 (page 69).
	3–8	Ending date and time	Six fields containing, respectively, the year, month, day, hours, minutes and seconds corresponding to the end of the simulation of the corresponding shower.
Real	1	Total number of shower particles	Total number of particles processed during the simulation of the corresponding shower.
	2	Total number of lost particles	Total number particles that went outside the region of interest for the simulations.
	3	Number of low energy particles	Total number of particles whose kinetic energies fell below the corresponding thresholds.
	4	Number of particles reaching ground	Total number of particles that reached the ground level, including also those particles not saved in the compressed file.
	5	Total number of unphysical particles	Number of "particles" generated by special procedures –like the splitting algorithm, for example– which cannot be associated with physical particles. This number is generally very small.
	6	Total number of neutrinos	Total number of neutrinos (ν_e, $\bar{\nu}_e$, ν_μ, $\bar{\nu}_\mu$) generated during the simulation of the current shower.
	7	Particles too near to the shower core	Number of particles that were *not* saved in the compressed file because they were too near to the shower axis (see text).
	8	Particles in the resampling region	Number of particles that were processed with the resampling algorithm (see text).

Table 4.2. *Fields contained in the "*end of shower*" record of compressed particle files. The structure of this record does not depend on the compile-time option selected for the particle record.*

	Field	*Name*	*Description*
Real	9	Particles too far from the shower core	Number of particles that were *not* saved in the compressed file because they were too far from the shower axis (see text).
	10	Shower maximum depth (Xmax) (g/cm2)	Vertical depth of the point where the number of charged particles is maximum, $X_{\max}$, obtained from a fit to the simulation data (see section 4.1).
	11	Total charged particles at shower maximum	Number of charged particles at $X_{\max}$, $N_{\max}$, calculated as explained in section 4.1 (page 69).
	12	Energy of lost particles (GeV)	Total energy of the particles of real field number 2.
	13	Energy of low-energy particles	Total energy of the particles of real field number 3.
	14	Energy of ground particles (GeV)	Total energy of the particles of real field number 4.
	15	Energy of unphysical particles (GeV)	Total energy of the particles of real field number 5.
	16	Energy of neutrinos (GeV)	Total energy of the particles of real field number 6.
	17	Energy lost in the air (GeV)	Total amount of energy lost by continuous medium losses (ionization losses) due to charged particles moving through the air.
	18	Energy of particles too near to the core	Total energy (in GeV) of the particles of real field number 7. This field is not defined for the longitudinal tracking particle file.
	19	Energy of resampled particles	Total energy (in GeV) of the particles of real field number 8. This field is not defined for the longitudinal tracking particle file.
	20	Energy of particles too far from the core	Total energy (in GeV) of the particles of real field number 9. This field is not defined for the longitudinal tracking particle file.
	21	CPU time (sec)	Amount of processor time required for the simulation of the current shower.

Table 4.2. *(continued)*

	Field	*Name*	*Comment*
Integer	1	Particle code	Stores the code of the corresponding primary particle.
Real	1	Energy (GeV) (log)	The logarithm of the corresponding primary particle's energy.
	2	Direction of motion (x component)	With respect to the AIRES coordinate system (see page 9).
	3	Direction of motion (y component)	
	4	Direction of motion (z component)	
	5	X coordinate (m)	Particle injection coordinate, with respect to the AIRES coordinate system (see page 9).
	6	Y coordinate (m)	
	7	Z coordinate (m)	
	8	Injection depth (g/cm2)	
	9	Injection time (ns)	
	10	Particle weight	Initial statistical weight of the corresponding particle.

Table 4.3. *Fields contained in the "*external primary particle*" record of compressed particle files. The structure of this record does not depend on the compile-time option selected for the particle record.*

	Field	*Name*	*Description*
Integer	1	Version of external module	User-settable integer in the range $[0, 759375]$.
Real	1	Total number of primaries	Total number of primary particles.
	2	Unweighted primary entries	Unweighted number of primary entries.
	3	Total energy of primary particles (GeV)	Total energy of primary particles (weighted).

Table 4.4. *Fields contained in the "*special primary trailer record*" of compressed particle files. The structure of this record does not depend on the compile-time option selected for the particle record.*

Arrival time. The saved quantity is the arrival time delay $t - t_0$, where t is the absolute time (measured from the beginning of the shower), and t_0 is the global time shift described in table 4.1 (page 76).

Particle weight. The statistical weight of the particle (see section 2.3).

Creation depth. The vertical atmospheric depth of the point where the particle was inserted into the simulating program's stacks.

Parent particle code. The code of the parent particle that generated (after a decay for example) the corrent one.

Parent particle energy. The logarithm of the kinetic energy of the parent particle.

Last hadronic depth. The vertical atmospheric depth corresponding to the last hadronic interaction suffered by the particle or by one of its ancestors.

Last hadronic projectile code. The particle code corresponding to the projectile of the last hadronic interaction suffered by the particle or by one of its ancestors.

Last hadronic projectile energy. The logarithm of the kinetic energy of the already mentioned projectile particle.

For each one of these quantities, a corresponding record field is defined. The complete list of fields is placed in table 4.5 (page 82). As mentioned previously, there are several record formats each one including a different subset of all the available fields.

In contrast with the beginning of shower and end of shower records, a given field of the particle record can be assigned different field numbers. As it will be seen below in this chapter, this does not affect the user's processing of compressed files, which can be done independently of the field number assignments.

There are specific IDL directives that can control the particles that are actually saved in the ground particle file.

To start with, let us consider the directives **RLimsFile** and **ResamplingRatio**, whose syntax is

RLimsFile grdpcles $r_{\min}$ $r_{\max}$
ResamplingRatio s_r

grdpcles identifies the file the directive refers to and $r_{\min}$ and $r_{\max}$ represent length specifications ($0 < r_{\min} < r_{\max}$). s_r is a real number ($s_r \geq 1$).

Such directives instruct AIRES to save *unconditionally* those particles whose distances to the shower axis lie within the interval $[r_{\min}, r_{\max}]$.

On the other hand, all the particles whose distances to the shower axis are smaller than

$$r_0 = \frac{r_{\min}}{s_r} \tag{4.3}$$

	Field				*Name*
	short	**normal**	**long**	**xlong**	
Integer	1	1	1	1	Particle code
	–	–	–	2	Parent particle code
	–	–	–	3	Last hadronic projectile code
Real	–	–	–	1	Parent particle energy (GeV) (log)
	–	–	–	2	Last hadronic proj. energy (GeV) (log)
	1	1	1	3	Energy (GeV) (log)
	2	2	2	4	Distance from the core (m) (log)
	3	3	3	5	Ground plane polar angle (radians)
	–	4	4	6	Direction of motion (x component)
	–	5	5	7	Direction of motion (y component)
	4	6	6	8	Arrival time delay (ns)
	5	7	7	9	Particle weight
	–	–	8	10	Particle creation depth (g/cm2)
	–	–	9	11	Last hadronic interaction depth (g/cm2)

Table 4.5. *Fields contained in the particle records of compressed ground particle files. The field numbers for the different particle records selectable at compilation time (see text), named* ***short****,* ***normal****,* ***long****, and* ***xlong*** *records, are tabulated. Notice that a given field can have different field numbers.*

(r_0 is, by definition, not larger than $r_{\rm min}$) are *not* included in the ground file, but their number and energy are recorded and the totals are included in the "end of shower record" (fields 7, 9, 18, and 20).

Finally, all the particles whose distances r to the shower axis lie in the interval $[r_0, r_{\rm min}]$ are processed by a *resampling algorithm* which *conditionally* keeps the particles accordingly with the following rules: (i) "Nonnumerous" particles –like pions, nucleons, etc.– are always saved. (ii) For every "numerous" particle –i.e., gammas, electrons, positrons and muons– in the mentioned region, the acceptance probability is[8]

$$p_s = \left(\frac{r}{r_{\rm min}}\right)^2 . \tag{4.4}$$

(iii) The statistical weights of the accepted particles are increased via

$$w' = \frac{w}{p_s}, \tag{4.5}$$

in order to keep unbiased the sampling algorithm.

The total number and energy of particles that fall within the resampling area, are recorded in the "end of shower record" (fields 8 and 19).

The **SaveInFile** (**SaveNotInFile**) directive permits including (excluding) one or more particle kinds into (from) the compressed file. Section 3.2.5 (page 39) contains several illustrative examples

[8]The expression of the acceptance probability is inspired in a suggestion by P. Billoir [37].

	Field						*Name*
	short	**norm (a)**	**norm (b)**	**long**	**x-long**	**x-x-long**	
Integer	1	1	1	1	1	1	Particle code
	2	2	2	2	2	2	Observing levels crossed
	–	–	–	–	–	3	Parent particle code
	–	–	–	–	–	4	Last hadronic projectile code
Real	–	–	–	–	–	1	Parent particle energy (GeV) (log)
	–	–	–	–	–	2	Last hadronic proj. energy (GeV) (log)
	–	1	–	1	1	3	Energy (GeV) (log)
	–	–	1	2	2	4	Direction of motion (x component)
	–	–	2	3	3	5	Direction of motion (y component)
	1	2	3	4	4	6	Particle weight
	2	3	4	5	5	7	Crossing time delay (ns)
	3	4	5	6	6	8	X coordinate (m)
	4	5	6	7	7	9	Y coordinate (m)
	–	–	–	–	8	10	Particle creation depth (g/cm2)
	–	–	–	–	9	11	Last hadronic interaction depth (g/cm2)

Table 4.6. *Fields contained in the particle records of compressed longitudinal tracking particle files. The field numbers for the different particle records selectable at compilation time (see text), named* ***short****,* ***normal (a)****,* ***normal (b)****,* ***long****,* ***extra-long****, and* ***extra-extra-long*** *records, are tabulated. Notice that a given field can have different field numbers.*

on how to use them.

Notice that by default, *all* particle kinds are enabled to be saved into the ground particle file.

Longitudinal tracking particle file

The structure of the longitudinal tracking particle file is very similar to the already described ground particle file: Both files have virtually the same "beginning of shower", "end of shower", "external primary particle", and "special primary trailer" records; and there are alternative formats for the particle records.

For that reason, it is highly recommended to the reader be familiar with the contents of the previous section describing the ground particle file before proceeding to read the present section. We shall limit here to briefly describe only those aspects that are somehow different in both files.

The longitudinal tracking particle file contains records storing detailed information about the particles that cross the defined observing levels. Since the observing levels are generally located at altitudes that include the shower maximum, and due to the fact that a single particle can cross more than one observing level during its life, it is clear that the longitudinal files can potentially be much larger than the average ground particle files.

For that reason, a special effort was made to save as much space as possible, and various record formats were defined to allow the user to select just the necessary fields. The record format selection can be done during installation, following the instructions placed in appendix A (page 101). Table 4.6 (page 83) lists all the defined data fields for the different default records.

The second integer field, named "Observing levels crossed" contains information about the observing levels the particle has crossed, and simultaneously about its direction of motion.

Let N_o ($N_o \leq 510$) be the number of defined observing levels. At a certain monitoring operation, a given particle crosses several observing levels from level i_f to level i_l (i_l may be equal to i_f). Let u_z be the z-component of the particle's direction of motion. If $u_z > 0$ ($u_z < 0$) the particle goes upwards (downwards), and therefore $i_f \geq i_l$ ($i_f \leq i_l$).

All this information is encoded in a single integer number called the *crossed observing levels key,* L, defined by the following equation:

$$L = i_f + 512\, i_l + 512^2\, s_{\rm ud} \tag{4.6}$$

where

$$s_{\rm ud} = \begin{cases} 1 & \text{if } u_z > 0 \\ 0 & \text{if } u_z \leq 0 \end{cases} \tag{4.7}$$

The three variables that appear in the right hand side of equation (4.6) can be easily reconstructed when L is known (see page 92).

The real fields listed in table 4.6 (page 83) are defined similarly to the corresponding ground particle record fields, with the exception of the x and y coordinates which are defined as follows:

Coordinates. The (x, y) coordinates are the Cartesian coordinates of the point where the particle crossed the level i_f, **measured from the intersection between the shower axis and the corresponding observing level's surface.**

Time delay. Defined as the difference $t - t_f$ where t is the particle's absolute time and t_f is the time required for a particle moving along the shower axis at the speed of light to go from the injection point to observing level i_f.

The IDL directives **RLimsFile**, **ResamplingRatio**, **SaveInFile** and **SaveNotInFile** can be used with longitudinal files to control when a particle must be saved or not. The last two directives do not present special difficulties, and work as explained in section 3.2.5 (page 39).

On the other hand, the directives[9]

RLimsFile `lgtpcles` $r_{\rm min}$ $r_{\rm max}$
ResamplingRatio s_r

define three parameters, $r_{\rm min}$, $r_{\rm max}$ and s_r, that are used to determine whether a particle record must be saved or not. The rules are the following:

[9]Notice that the parameter controlled by directive **ResamplingRatio** is *global,* that is, its last setting applies to every one of the compressed files in use.

1. Let $X_c = 0.8\,X_i + 0.2\,X_g$, where X_i and X_g are the vertical injection and ground depths, respectively.

2. For each observing level i, $i = 1, \ldots, N_o$, let

$$r_i = \begin{cases} 0 & \text{if } X_o^{(i)} \le X_c \\ \left(\dfrac{X_o^{(i)} - X_c}{X_g - X_c}\right) r_{\min} & \text{if } X_c < X_o^{(i)} < X_g \\ r_{\min} & \text{if } X_o^{(i)} \ge X_g \end{cases} \tag{4.8}$$

   where $X_o^{(i)}$ is the vertical depth of observing level i; and let $r_{0i} = r_i/s_r$.

3. Any particle crossing observing levels will *not* be saved in the longitudinal file if one of the following conditions is true:

   (a) $|x| < r_{0i}$ **and** $|y| < r_{0i}$.

   (b) $|x| > r_{\max}$ **or** $|y| > r_{\max}$.

   x and y are the Cartesian coordinates of the particle at observing level i_f, measured from the intersection between the shower axis and the corresponding observing level.

4. Gammas, electrons, positrons and muons crossing observing levels and verifying the two following conditions

   (a) $|x| < r_{\min}$ **and** $|y| < r_{\min}$.

   (b) $|x| > r_{0i}$ **or** $|y| > r_{0i}$.

   (in the same notation of the previous point), will be *conditionally* kept, with probability and reweighting factor given by equations (4.4) and (4.5), respectively.

5. All the particles not fulfilling the conditions of the preceding points will be *unconditionally* saved in the file.

These rules set varying limits for the zone of excluded particles. In the zone near the shower axis, all particles crossing observing levels placed above X_c will be saved, then the exclusion zone enlarges proportionally to the depth of the observing level, reaching the value indicated in the **RLimsFile** directive at the ground depth. Notice that X_c divides the complete shower path (as measured in atmospheric depth) into two, upper-lower, 20%-80%, zones.

The number of defined observing levels affects the degree of detail of the monitoring of the longitudinal shower development, and some applications usually require that this number be as large as possible. On the other hand, such setting may lead to the generation of very big longitudinal particle files since a large number of data records are generated as long as every particle crosses the observing levels. To overcome this difficulty, AIRES includes a selection mechanism to avoid including in the

compressed file the information related with all the defined observing levels. Consider the following illustrative example:

```
ObservingLevels 100
SaveInFile lgtpcles e+ e-
RecordObsLevels None
RecordObsLevels 1
RecordObsLevels 4
RecordObsLevels 10 90 10
RecordObsLevels Not 20
```

The first directive sets the number of observing levels to 100, and the second one enables particle saving in the longitudinal particle tracking file. In this case only electrons and positrons will be recorded (Notice that the longitudinal file is disabled by default, and therefore it is necessary to use unless one **SaveInFile** instruction to enable it). The default action is to record particles crossing *any* of the defined observing levels, and the remaining instructions are placed to override this default setting. The directive **RecordObsLevels None** eliminates all the defined observing levels from the set of levels to be taken into account to save particle records into the compressed file. The actions of the instructions that follow are, respectively: Mark level 1 for recording particles crossing it; idem level 4; idem all levels from 10 to 90 in steps of 10 levels; unmark level 20. The resulting set of marked levels is $\{1, 4, 10, 30, 40, 50, 60, 70, 80, 90\}$.

4.2.2 Using the AIRES object library

The *AIRES object library* is a set of routines designed with the main purpose of providing adequate tools to analyze the data saved in the compressed output files.

Appendix D (page 140) explains in detail the contents of the library and how to use it. In this section some illustrative examples are presented.

From now on we are going to assume that the AIRES file is being processed by a program, provided by the user and similar to the demonstration programs that are included with the AIRES software distributions.

We are going to use FORTRAN in our examples, but this is not a restriction since the AIRES library includes routines for a *C interface,* which allow the C user to fully exploit the library's resources.

Output particle codes

Every analysis program must begin with a call to routine **ciorinit**. This routines sets up the environment where the library routines can work adequately.

This routine permits setting the particle coding system that the user wants to work with. It is possible to select either the AIRES coding system already described in section 2.2.1 (page 17), or other usual coding systems. The coding systems known by AIRES 19.04.08 are the following:

1. Aires internal coding.

Particle	***Codes***					
	AIRES	***PDG***	***CORSIKA***	***GEANT***	***SIBYLL***	***MOCCA***
γ	1	22	1	1	1	1
e^+	2	−11	2	2	2	2
e^-	−2	11	3	3	3	−2
μ^+	3	−13	5	5	4	3
μ^-	−3	13	6	6	5	−3
τ^+	4	−15	86	86	20	10
τ^-	−4	15	87	87	−20	−10
ν_e	6	12	66	4	15	0
$\bar{\nu}_e$	−6	−12	67	−	16	0
ν_μ	7	14	68	4	17	0
$\bar{\nu}_\mu$	−7	−14	69	−	18	0
ν_τ	8	16	4	4	−	0
$\bar{\nu}_\tau$	−8	−16	−	−	−	0
π^0	10	111	7	7	6	5
π^+	11	211	8	8	7	4
π^-	−11	−211	9	9	8	−4
K^0_S	12	310	16	16	12	12
K^0_L	13	130	10	10	11	13
K^+	14	321	11	11	9	11
K^-	−14	−321	12	12	10	−11
η	15	221	17	17	22	14
n	30	2112	13	13	14	6
$\bar{n}$	−30	−2112	25	25	−14	−6
p	31	2212	14	14	13	7
$\bar{p}$	−31	−2212	15	15	−13	−7

Table 4.7. *Elementary particle codes corresponding to several commonly used coding systems, for the most relevant particles. The routines that process AIRES compressed output files allow the user to select any one of these coding schemes.*

2. Aires coding for elementary particles and decimal nuclear codes ($A + 100\,Z$).
3. Particle Data Group coding system [38], extended with decimal nuclear codes ($A + 10^5\,Z$).
4. CORSIKA simulation program particle coding system [36].
5. GEANT particle coding system [39].
6. SIBYLL [10] particle coding system, extended with decimal nuclear codes ($A + 100\,Z$).
7. MOCCA-style particle codes [1], extended to match all AIRES particles.

The codes corresponding to elementary particles are listed in table 4.7.

Opening existing files

Once the proper environment is set up by means of the initializing routine, the system is ready to open any existing compressed file. The open routine **opencrofile** will use the header information to initialize the internal variables that permit processing the different fields defined for the file. The following example illustrates how to open a file:

```
program sample
character*80 mydir, myfile
integer channel, irc
. . .
call ciorinit(0, 1, 0, irc)
. . .
call opencrofile(mydir, myfile, 0, 10, 4, channel, irc)
. . .
```

myfile and **mydir** are character strings containing respectively the file name and the directory where it is placed. The integer argument "10" indicates that the logarithmic fields are going to be transformed into decimal logarithms. **channel** is an output parameter identifying the opened file; it should not be set by the calling program.

It is important to remark that this call will transparently open **any** compressed file, regardless of its type or format (ground particle as well as longitudinal tracking particle files in all their variants), the AIRES version used to write it and/or the machine used when writing it.

Getting information about the file

The headers of the compressed files are divided into two parts: One part containing the definitions of the file's data records and another section with information about the simulations that originated the file.

The file definitions are specific to each opened file, and therefore the system must store them separately for each one of the files that are simultaneously open.

The other information, however, is of global character, and so the available data always corresponds to the last opened file. These data are superseded each time **opencrofile** is called.

Routine **croheaderinfo** prints a summary of this global information while **croinputdata0** copies some of those data into arrays to make them available to the user (see page 152) and **crotaskid** returns task name information. Functions **getinpint**, **getinpreal**, **getinpstring** and **getinpswitch** (see pages 176–179) allow to obtain other input data items not returned by **croinputdata0**. **getglobal** can be called to retrieve information about global variables that were defined during the simulations. **idlcheck** returns information about the IDL instructions that were valid when the file was generated, and **crofileversion** and **thisairesversion** return version information that might be useful when reading compressed files written with old AIRES versions.

In some special applications it is necessary to access information that can be stored only in the internal dump file. In these cases, it can be helpful to invoke the routine **loadumpfile** right after opening the corresponding compressed file, and then use some of the routines **dumpinputdata0**, **dumpfileversion**, etc., to access the mentioned data.

The structure of any already opened file can be printed calling routine **crofileinfo** which prints a list of the different records defined for the corresponding file and the names of the fields within records. It is also possible to load into arrays such information by means of routine **crorecstrut** in order to make it available to the analysis program.

Reading the data records

Once a file is open, it remains positioned at the beginning of the compressed data section. From then on, the file can be sequentially read using routine **getcrorecord**:

```
okflag = getcrorecord(channel, indata, fldata, altrec,
                      0, irc)
```

getcrorecord returns logical data, which in this case are stored in the logical variable `okflag`. The returned value is "**true**" if the reading operation was completed successfully, "**false**" otherwise (end of file, I/O error, etc.).

ciochann should be the same integer variable used when opening the file; it identifies the file to be processed.

irc is an integer return code. If **okflag** is "false", then the return code contains information about the error that generated the abnormal return, as explained in page 172. For successful read operations, **irc** indicates the record type that has been just read in: 0 for the default particle record, 1 (2) for the "beginning (end) of shower" record, etc. At the same time, the logical variable **altrec** distinguishes between "alternative" (non default) records (**true**), from default ones (**false**).

The data stored in the different fields of the record is retrieved by means of the arrays **indata** and **fldata**. Both are single index arrays, containing integer and double precision data, respectively. The data items stored in these arrays does vary with the kind of file being processed and the type of record

that was scanned. In all cases, the routine will automatically set the relevant elements of these arrays accordingly with the logical definition of the record, regardless of the physical structure of it which remains absolutely hidden at the user's level.

To fix ideas, let us suppose that a ground particle file with *normal* particle records is being processed. Every time **irc** is zero (default record), the integer and real data arrays will contain the elements listed in table 4.5 (page 82), that is

```
indata(1)  ←  Particle code

fldata(1)  ←  Energy (GeV) (log)
fldata(2)  ←  Distance from the core (m) (log)
fldata(3)  ←  Ground plane polar angle (radians)
.   .   .
fldata(7)  ←  Particle weight
```

For different return codes, the number of assigned array elements may be different, as well as their meanings; but in all cases such data items will be set accordingly with the corresponding record sequence (tables 4.1, 4.2, etc.).

In order to make the analysis programs simpler and more robust, a special routine has been included in the AIRES library to *automatically* set the adequate field indices corresponding to a given record of a certain compressed file, as illustrated in the example of figure 4.2.

The outstanding characteristic of this piece of code is that the elements of arrays **indata** and **fldata** are not referenced directly using numeric indices, but by means of integer variables like **icode** for instance (see figure 4.2).

Those index variables are set by means of routine **crofieldindex**. The arguments required by this routine include: (i) The identification of the file (**channel**). (ii) The record type, coincident with the return codes of **getcrorecord** already mentioned. In this example 0 for the default record and 2 for the "end of shower" record. (iii) The first characters of the field name. Fields are identified by their *names,* providing therefore absolute transparency to the fact that the order and number of fields may change with the file being processed. The next argument of **crofieldindex** is set to 4 to force the program to stop in case of ambiguous or erroneous field specification, thus providing a very safe processing environment. (iv) The output argument **datype** returns information about the data type corresponding to the specified field, as explained in page 147. In the particular case of longitudinal particle tracking files, it is generally convenient to use the routine **getlgtrecord** in place of **getcrorecord**. A complete description of **getlgtrecord** can be found in page 181; this routine must be used jointly with **getlgtinit**.

Closing files and ending a processing session

The AIRES library routines support simultaneous processing of more than one compressed file. Several compressed files can be opened at the same time, each one identified by the corresponding channel integer variable.

```
program sample
. . .
integer datype, irc, icode, idist, inear
integer indata(30)
double precision fldata(30)
integer particlecode
double precision logdistance, numberofnear
. . .
call ciorinit(0, 1, 0, irc)
call opencrofile(mydir, myfile, 0, 10, 4, channel, irc)
. . .

icode = crofieldindex(channel, 0, 'Particle code',
                      4, datype, irc)
idist = crofieldindex(channel, 0,
                      'Distance from the core',
                      4, datype, irc)
inear = crofieldindex(channel, 2, 'Particles too near',
                      4, datype, irc)


. . .

okflag = getcrorecord(channel, indata, fldata, altrec, 0,
                      irc)


if (irc .eq. 0) then
  particlecode = indata(icode)
  logdistance  = fldata(idist)
  . . .
else if (irc .eq. 2) then
  numberofnear = fldata(inear)
  . . .
end if
. . .
```

Figure 4.2. *Processing compressed data files, an example illustrating how to set field indices automatically.*

The opened files can be closed using two alternative procedures (see page 144): (i) Routine **cioclose1** closes individual files. **cioclose** closes *all* the currently opened files.

Routine **cioclose** should be used only if the processing session will continue after closing all files. To finish an analysis program in an ordered fashion use the routine **ciorshutdown**. This procedure performs all the required tasks to properly set down the processing system, including a call to **cioclose**.

Other operations

There are many other routines included in the AIRES library that provide useful tools for special analysis tasks. Such routines are explained in detail in appendix D (page 140), we shall limit here to a brief presentation of the most relevant ones:

Counting records. Routines **crorecinfo** and **croreccount** count the data records contained within a compressed file.

Repositioning. Routine **crorewind** repositions an already opened file at the beginning of the data section. Routines **crorecnumber** and **crogotorec**, used jointly, permit accessing the data records in arbitrary order.

Fast scanning of a file. Routine **crorecfind** finds the next appearance of a record of a given type (to locate shower headers, for example). **getcrorectype** returns the type of the next record, and **regetcrorecord** re-reads the current record.

Longitudinal tracking file utilities. Routine **crooldata** returns basic information about the positions of the observing levels defined for the simulations; while **olcoord** returns the coordinates of the intersections between the observing levels and the shower axis and **olv2slant** evaluates the slant depths corresponding to each observing level. Routines **olcrossed** and **olcrossedu** decode the crossed observing levels key defined in equation (4.6), returning the variables i_f, i_l and $s_{\rm ud}$ (see section 4.2.1). The logical function **olsavemarked** permits determining whether or not a given observing level is recorded into a compressed file.

Special primary utilities. Besides the specific routines designed to process special primary particles, described in detail in section 3.5, the AIRES library includes also some auxiliary routines that are useful to obtain data about the special primaries that were defined at the moment of creating the compressed file that is being analyzed. **crospcode** and **crospmodinfo** are examples of such procedures.

Miscellaneous routines. The library contains some other routines than may be useful in certain applications, for example the pseudo-random number utilities **raninit**, **urandom**,**urandomt**, and **grandom**; Gaisser-Hillas function related routines: **fitghf**, **ghfpars**, **ghfx**, **ghfin**; atmospheric depth utility routines line **xslant**; etc.

The AIRES object library is continously evolving, so additional procedures will be surely included in future AIRES versions.

Chapter 5

The AIRES Runner System

Production simulation tasks usually require large amounts of computer time to complete, and in such cases the user risks loosing all the simulation run if the system goes down before the task is finished. To avoid this inconvenient situation, the AIRES simulation system provides a special auto-saving mechanism that permits splitting the simulation job into small runs. In case of abnormal interruption, the simulations can be restarted at the point they were when the last auto-saving was performed.

As explained in chapter 3 (page 34), a simulation task may require several invocations of the simulation program if the auto-save mechanism is enabled. If this is done manually, the user must control the sequence of instructions needed to complete the simulations. To ease the management of such sequential series of processes, a set of scripts were developed with the capability of automatically launching the corresponding jobs. These scripts are part of the *AIRES Runner System (ARS)*, designed as a set of interactive procedures to manage complex simulations tasks.

The AIRES Runner System works only on UNIX platforms, and provides tools for input file checking, sequential and concurrent task processing, event logging, etc. This chapter is devoted to present some examples that will help the user to get familiar with the Runner System.

There are many parameters that modify the behavior of the AIRES Runner System. Most of them are user-settable and their definition statements are placed within the ARS initializing file **.airesrc**. In standard AIRES installations, this file is placed in the user's home directory.

5.1 Checking input files

In section 3.2.2 (page 35), the IDL directives **CheckOnly** and **Trace** were used to instruct AIRES simulation programs to scan a given input file, report on its contents and stop without actually starting the simulations.

The ARS command

```
airescheck -t myfile.inp
```

will invoke **Aires** with the same input as displayed in page 36. The **-t** qualifier is placed to enable typing the input lines as long as they are scanned.

There are additional qualifiers accepted by this command, for example:

```
airescheck -tP -p AiresQIIr04 myfile.inp
```

The **-p** qualifier overrides the default simulation program used to process the input file, and the **P** switch indicates that the output must be printed instead of being typed at the terminal. The print command to use can be set modifying the **.airesrc** initializing file.

5.2 Managing simulation tasks

Once the input file has been checked, the simulations can be started. The command

```
airestask myfile
```

will first check that file **myfile.inp**[1] exists, and then will create an entry in the corresponding *ARS spool*. Finally, **aireslaunch** will be executed.

The **aireslaunch** script will detect that there is a task pending completion and so will prompt the user to start the simulations. In case of positive answer, the simulation program will be started with the corresponding input, and will be repeatedly invoked if necessary until the task is completed[2]. All those operations are completely automatic, no further user intervention is normally required.

If there are more than one task to be processed, they can be spooled at any moment after launching the first simulations. The command

```
airestask my_other_file
```

will make a new spool entry which will be queued after the first one. Execution of this task will start as soon as the previous one is finished. There is no limit in the number of tasks that can be queued in the ARS spools.

At any moment during the simulations, it is possible to inspect the evolution of the spooled tasks by means of the ARS command **airesstatus**.

In the preceding examples, the default simulation program (which normally is the **Aires** program) will be used. There are two alternatives to override the default specification: (i) Modify the default program setting of the initialization file **.airesrc**. (ii) Use the **-p** qualifier of the **airestask** command:

```
airestask -p AiresEPLHC yet_another_file
```

AiresEPLHC is the name of a variable defined within the initialization file, which indicates the executable program that contains a link to the EPOS LHC hadronic package.

[1] **airestask** first assumes a default extension **.inp** for the input file name, and as a second alternative, tries to find the file whose *complete* name is as specified in the input parameter.

[2] The simulation program communicates with the script via a file that contain information about the status of the simulations.

5.2.1 Canceling tasks and/or stopping the simulations

Every spooled task can be canceled by means of the command **airesuntask**, for example:

```
airesuntask my_other_file
```

will erase the second spooled task of the preceding section. If the **airesuntask** command is invoked with no parameters, then it will prompt the user to cancel each one of the spooled tasks.

It is not recommendable to remove the spool entries corresponding to tasks that are currently running. In such cases it is better to first stop the simulation program, and wait until the AIRES Runner System shuts down.

The simulation program can be stopped with the ARS command **airesstop**, which generally is invoked with no arguments. This script originates an ordered shutdown of the simulations, which includes an update of the internal dump file, and may take up to several minutes to effectively interrupt the simulations. The command **airesstatus** can be used to monitor the status of the system during this process.

On the other hand, a currently running simulation can be immediately aborted by means of command **aireskill**. In this case the corresponding processes are killed without any previous auto-saving operation.

Stopped simulations can always be restarted using **aireslaunch**.

5.2.2 Performing custom operations between processes

Every time a process[3] ends, the ARS checks for the existence of a executable script named **AfterProcess** (case sensitive!), first in the current working directory, and then –if not found– in the default directory of the user's account (HOME directory). If the file is found, it is executed.

The complete command line used when invoking the **AfterProcess** macro is the following:

```
AfterProcess  spool tn msg rc trial totsh lastsh prog
```

where

spool is the spool identification.

tn is the task name.

msg is a message string coming from the simulation program. Normally it takes the values **EndOfTask** or **EndOfRun**, indicating if the current task was or not finished, respectively. Other values are also possible and correspond to abnormal situations.

rc is a numeric parameter, taking the value 2 if the run has been stopped using an AIRES.STOP file (command **airesstop**).

[3] See section 3.1 (page 34).

trial is a numeric variable counting the number of trials for the current run. Generally takes the value 1, but in certain circumstances, for example when relaunching AIRES after a system crash, it can take larger values.

totsh is the total number of showers for the current task.

lastsh is the last completed shower.

prog is the instruction used to invoke AIRES, which includes the full name of the simulation program used in the last run.

This powerful ARS option makes it possible for the user to perform operations of almost every kind after ending the processes. Of course, a certain degree of expertise with UNIX systems may be required in certain cases. Typical examples of operations that can be done using this facility are: File movement after completion of tasks (for example to massive storage systems), alerts of any type about conditions of the system, like full disks, etc.

On return, the **AfterProcess** script can communicate with the ARS via the exit code. If it is zero then processing will continue normally, otherwise the ARS will send a mail notifying the abnormal return code and then will stop. If it is necessary to restart the simulations, it can be done using the ARS command **aireslaunch**.

The following shell script is a very simple example of an "after process" macro:

```
#!/bin/sh
#
if[ $3 = EndOfTask ]
then
#
# This code will be executed only after ending a task.
#
  mv ${2}.grdpcles /mysafeplace
fi
exit 0
```

Notice that no action will be taken up to the end of a task. Whenever this happens, the corresponding ground particle file is moved to another directory. The command **exit 0** ensures normal return code; **exit *n*** with $n \neq 0$ means an abnormal exit and in this case the simulations will be stopped.

Similarly as in the case of the **AfterProcess** macro, the ARS system will search for a **BeforeProcess** macro, right before invoking the simulation program. The existence of the **BeforeProcess** macro is checked in the working directory, and in the user's account (HOME) directory (in that order). If the file is found, it is executed.

The complete command line used when invoking the **BeforeProcess** macro is the following:

```
BeforeProcess  spool tn trial ifile prog
```

where

spool is the spool identification.

tn is the task name, or **UNKNOWN** if the task is not initialized yet.

trial is a numeric variable counting the number of trials for the current run. Generally takes the value 1, but in certain circumstances, for example when relaunching AIRES after a system crash, it can take larger values.

ifile is the name of the input file to be used when running the simulation program.

prog is the instruction used to invoke AIRES, which includes the full name of the simulation program used in the next run.

After completing execution of the **BeforeProcess** macro, the ARS checks the corresponding return code, continuing with the next step only if it is zero.

5.3 Concurrent tasks

In many cases it is necessary to simultaneously process more than one task. Systems having more than one CPU and/or clusters of machines sharing the same file system, are examples of such situation.

The AIRES Runner System provides certain tools designed to work under such circumstances. The key idea is to define more than one spool, and assign one spool to each processing unit, either a CPU or a machine inside the cluster.

In the preceding examples, the **airestask** command was invoked without spool specification. The default spool is used in case of missing specification, and that is what was actually done in those examples.

In the standard configuration there are 9 predefined spools, named respectively "1", "2", ..., etc. Spool "1" is the default spool[4]. The command

```
airestask -s 2 myfile
```

will create a spool entry placed in spool "2". The user will be prompted to start the simulations if there is currently no activity related with that spool. The command

```
airesstatus 2
```

[4]The ARS includes also the commands **mkairesspool** and **rmairesspool** which allow the user to respectively create and delete spool directories.

will report on the simulations that are running at spool "2". Similarly,

```
aresstatus all
```

will report on the simulations that are running at every active spool.

In the following interactive session, it is illustrated how to launch three simultaneous tasks (it is assumed that the machine possesses various CPU's which can be automatically assigned to the launched processes):

```
cd directory1
aireslaunch -s 1 task1
. . .
cd directory2
aireslaunch -s 2 task2
. . .
cd directory3
aireslaunch -s 3 -p AiresQIIr4 task3
. . .
```

It is most important that the working directories of different tasks be also different: Concurrent simulation programs running with the same working directory may generate conflicts when communicating with the ARS scripts. This fact is stressed by means of the **cd** commands of the example, where **directory1**, **directory2** and **directory3** *must* be different directory specifications.

Notice also that the third spooling command makes use of an alternative simulation program in order to perform a different kind of simulation. Alternative programs may also be necessary when running simulations on clusters sharing the same file system but made with non compatible platforms. In those cases it is necessary to have different executable modules for each platform. Once such modules are available, it is possible to change the default programs corresponding to the different spools by means of suitable modifications to the **.airesrc** initialization.

The details about how to make the AIRES Runner System work in complex operating environments are rather technical and go beyond the scope of this manual. Such a job requires normally a good degree of expertise on UNIX systems.

5.4 Some commands to manage dump file data

Chapter 4 (page 69) explains in detail the operations needed to retrieve data stored within the internal dump file in either its binary or ASCII versions. Some of them are frequently used and generally involve very similar sequences of instructions. A typical example is to export one or more tables corresponding to an already finished task.

The ARS includes a shell script that can be helpful in those cases. Consider for example the command (under UNIX)

```
airesexport mytask 1001 1205 to 1213
```

Its action is to invoke the AIRES summary program with the following input

```
Summary Off
TaskName mytask
ExportTable 1001
ExportTables 1205 1213
End
```

generating text files for tables 1001, 1205, 1207, 1211 and 1213 (see appendix C).

In some cases it may be necessary to specify other parameters, like in the following example

```
airesexport -w idfdir -O LM -s mytask 2501
```

This command will generate single shower tables (enabled by the **-s** qualifier) as well as average ones. The options **LM** correspond to $dN/d\log_{10}E$ distributions with energies expressed in MeV (see section 4.1.2), and the string following the **-w** qualifier (**idfdir**) indicates the directory where the IDF and/or ADF files are located (The global directory accordingly with the definitions of section 3.3.2).

5.4.1 Converting IDF binary files to ADF portable format.

ADF files were implemented for AIRES version 2.0.0, and to have them written by the simulation programs after a task is completed, it is necessary to explicitly enable them by means of the IDL directive **ADFile**. The (binary) IDF is *always* generated, regardless of the input settings and/or the version of AIRES used.

Of course, the IDF stores all the data associated with both input parameters and output observables, and is enough for any kind of analysis provided the user always works with compatible computers. But this may not be the case when a person or group is working at different locations. For such cases, a *portable* file format is needed and the ADF becomes essential to enable data analysis in non-compatible workstations.

If the ADF was not generated during the simulations, or if the simulations were performed using a version of AIRES previous to version 2.0.0, it must be created manually. The current AIRES distribution includes an IDF to ADF converting program, whose default name is **AiresIDF2ADF**.

This program can be used directly. It is just necessary to invoke it (no arguments needed) and answer to the prompts that will be appearing.

On the other hand, the ARS includes a special shell script that permits converting files without calling **AiresIDF2ADF** manually. Let us illustrate how to use this command with an example. Suppose in a certain place there are some IDF files that need to be converted to ADF format. The UNIX command

```
idf2adf taskname1 taskname2 taskname3
```

will search for the files **taskname1.idf**, **taskname2.idf**, etc., and will call **AiresIDF2ADF** as many times as necessary, to create the portable files **taskname1.adf**, **taskname2.adf**, etc. Of course, the old IDF files will remain unchanged.

This script will work well in most cases. However, there might be special situations where it is necessary to use **AiresIDF2ADF** manually, for example when the IDF file is renamed with a new name not ending with ".idf".

Appendix A

Installing AIRES and maintaining existing installations

As mentioned in section 1.2 (page 8), every AIRES distribution is currently packed in a single compressed UNIX tar file. In this appendix it is assumed that the software distribution was successfully decompressed and tar expanded.

A.1 Installing AIRES 19.04.08

In UNIX platforms, the installing procedure is quite simple: Almost everything is done automatically. The key points to take into account are:

(a) A Unix shell script **doinstall** is provided. This script will install the software automatically.

(b) The file **config** contains all the customizable variables. You should check its contents and edit it if needed before invoking **doinstall**.

(c) There will be two main directories:

1. *Aires root directory* (hereinafter named **Aroot**), which is the highest level directory for the installed files. Normally the AIRES distribution compressed tar file and the AIRES External Input data compressed tar files are located in this directory before starting the installation process. You might need to specify **Aroot** by editing the **config** file (located within the **Iroot** directory). For standard, personal installation, the default (creating a directory named **aires** in your home directory) will be OK.
2. *Installation root directory* (hereinafter named **Iroot**), which is the directory where the distribution file was downloaded (that is, the directory containing the **doinstall** script).

Notice that the **Iroot** and **Aroot** directories may or may not be the same directory (Do not worry about this: The installation program will manage every case properly.).

(d) Your account must have access to a FORTRAN compiler (normally, command **gfortran**), and in some cases to a C compiler (commands **gcc**, etc.); and these compilers must be placed in one of the PATH directories (in other words, if you type at your terminal, say, **`gfortran`**, the machine will take **`gfortran`** as a known command). If the compilers are not in the PATH you will have to enter their absolute location manually in the **config** file (Our recommendation, however, is to ensure that the compilers are in the PATH. It is something not difficult to achieve. If you do not know how to proceed or what we are speaking about, then ask your local UNIX expert).

A.1.1 Installation procedure step by step

1. Ensure that you have write permission on both **Iroot** and **Aroot** directories, and in all their sub-directories.

2. **cd** to **Iroot**, and edit the file **config** if necessary. The current AIRES installation program will automatically set most of the required installation parameters, including automatic detection of the operating system (Linux and Mac OSX are the ones currently supported).

3. If you need a full installation that will allow you to run simulation tasks, you must also download and place inside the **Aroot** directory the ***External Input data file*** (available from the AIRES repository). The size of this file is about 300 MB.

4. Enter the command

   `doinstall 0`

   if you are installing AIRES for the first time, or

   `doinstall 1`

   if you are upgrading your current installation (This is the case for those users that are already employing a previous version of AIRES. Note that you **should not erase** any existing installation of AIRES before completing the upgrade.).

   This procedure will install the software using the data you set in step 2. This may take some minutes to complete. A message will be typed at your terminal indicating whether the installation was successful or not. If you get any error message(s), you should check all the requirements described previously, in particular points (d) and (1). Try also modifying the **config** file.

5. Type the command (case sensitive)[1]

   `Aires`

[1] The name **Aires** can be changed modifying adequately the **config** file. If this name was changed, then the user supplied name must be typed in place of the default one.

to see if the program is running and is in your search path. You should see typed at your terminal something like the following text[2]:

```
>>>>
>>>> This is AIRES version V.V.V (dd/Mmm/yyyy)
>>>> (Compiled by . . . . .
>>>> USER: uuuuu, HOST: hhhhhhh, DATE: dd/Mmm/yyyy
>>>>

> dd/Mmm/yyyy hh:mm:ss. Reading data from standard input unit
```

where **V.V.V** indicates the current version of AIRES (19.04.08) and goes together with the release date. Type **x** and press ⟨ENTER⟩ to leave the program.

If step 4 ended successfully and you fail to run the program, it is likely that the AIRES **bin** directory is not in your environment search path (Unix environment variable PATH). In some systems you need to log out and log in again to make effective any PATH change. If you cannot place the AIRES **bin** directory into your account's PATH, then ask a Unix expert to do that for you. Once you are sure that the directory is in the search path, and if the problem still persists, check if the executable file **Aires** exists. If it does not exist that means that step 4 was not successfully completed. Do not continue with the next step until you succeed with this one.

6. **Installation of extensions.** The current version of AIRES permits the installation of extensions that add new capabilities to the original system. At present, there is one of such extensions available: ZHAireS, that allows to simulate the radio wave emission that takes place during shower development (see reference [42] and the corresponding manual for details). Installation of extensions is not mandatory; if you do not need to do so you can skip this point and go directly to (7).

   To install the extension(s):

   (a) Download the extensions available (compressed tar files of the form ***Name*-Exten-*v*-*v*-*v*.tar.gz**), and place then in the same directory where the current AIRES distribution tar file is.

   (b) **cd** again to the **Iroot** directory. Together with the **doinstall** script that you have already used to install AIRES, you will find another script: **addextensions**.

   (c) If you want to perform a quick installation of the extensions, using the same installation parameters that are in the **config** file, just execute

```
addextensions
```

[2] You should also obtain a similar output if you invoke the AIRES/EPOS/QGSJET/SIBYLL simulation program instead of the default **Aires**.

This command will do all the necessary to install the extensions, including compilation of sources and building executable programs. Then go directly to (6e).

(d) Instead, if you need to customize the configuration parameters that are relevant to the extension, then execute

```
addextensions noinstall
```

This command will just expand the corresponding tar file and perform some checks, without compiling and/or building any program or library. After this is complete, for each installed extension you will see the corresponding **config.*extension_name_in_lowercase*** file. You can edit this file manually, perform all the needed changes, and complete the installation of the extension using the command

```
doinstall 3 extension_name_in_lowercase
```

(e) Follow the instructions placed in the specific **Install.*Extension*.HowTo** file to ensure that the corresponding program is working properly.

7. **cd** to your HOME directory and verify the presence of a file named **.airesrc**.

   Normally it is not necessary to change anything in this file, but the need may appear in the future, specially if you decide to use the UNIX scripts that are provided to help running AIRES (see chapter 5).

8. If you completed successfully these steps, the software should be properly installed.

9. After successfully completing these steps you can optionally delete the files corresponding to old versions of AIRES. Such files are placed within the Aroot directory. For example, directory 18-09-00 contains AIRES 18.09.00 files, etc.

10. If you completed successfully these steps, the software should be properly installed. In that case, and if you do not have previous experience using AIRES, we strongly recommend you to go to the Iroot directory again, enter the doc sub-directory, print the file LearnByExamples.txt, and follow the instructions that are in this file to learn how to use AIRES.

A.2 Recompiling the simulation programs

In many cases it may be necessary to recompile the simulation programs after having successfully installed the AIRES system. Some examples of such situations are:

- Some compilation parameters were not set accordingly with the user needs; or the required configuration is no more the one set up at the moment of installing the software.
- It is necessary to install AIRES in different (not compatible) platforms sharing the same directory tree.

- It is necessary to create more than one executable program, each one compiled with different compilation parameters. As an example of this case, consider that the number and kind of records that are written in the compressed particle files can be controlled by means of compilation parameters (see section 4.2.1), and that it is required to have the executables for different file formats.

The arguments recognized by the **doinstall** executable script allow the user to easily perform the different operation required in cases like the ones previously enumerated.

The general syntax of **doinstall** is:

```
doinstall ilev [ cfext ]
```

ilev is an integer ranging from 0 to 4 indicating the "level" of installation:

0 Complete installation of the AIRES system. Necessary only when installing AIRES for the first time.

1 Upgrade of an existing installation, making the installed version the new current version.

2 *Recompiling.* All the simulation programs and the summary program are compiled and linked. The AIRES object library is rebuilt.

3 *Relinking.* New executables for all the simulation programs and the summary program are created using the existing object files.

4 *Rebuilding the library.* The AIRES object library is rebuilt using the existing object files.

cfext is an optional argument. It is a character string indicating that an alternative configuration file must be used to set the installation parameters. If ***cfext*** is no null, then the file **config.*cfext*** is used instead of the default ***config*** file used when ***cfext*** is not specified.

To perform different compilation/installation jobs, it might be useful to have several configuration files. For example, the ***config*** file is first copied to a new **config.short** file. Then **config.short** is edited changing the following parameters: (i) The format for both ground and longitudinal tracking compressed files is set to "short". (ii) The name of the executable program **Aires** is changed into **Aires_sht**. Finally the command

```
doinstall 2 short
```

is executed. This will generate several new executable programs, namely, **Aires_sht**, **Aires_shtS23**, etc., which will be capable of producing compressed files with short format particle records.

Appendix B

IDL reference manual

All the main simulation programs and the summary program **AiresSry** use a common language to receive the user's instructions. This language is called *Input Directive Language* (IDL), and currently consists of some 70 different instructions to set simulation parameters, control the output data, etc. In this section we list, alphabetically ordered, all AIRES 19.04.08 IDL directives.

The IDL directives can be written using no special format, with one directive per line (there are no "continuation lines", but each line can contain up to 176 characters). The directives start with the directive name followed by the corresponding parameters. All the "words" that form a sentence must be separated by blanks and/or tab characters.

All directives are scanned until either an **End** directive or an end of file is found. Most directives can be placed in any order within the input stream. The **Input** directive permits inserting instructions placed in separate files letting the user to conveniently organize complex input data sets. **Input** directives can be nested.

Dynamic (can be set every time the input file is scanned), *static* (can be set only at task initialization time) and *hidden*[1] (associated with rarely changing parameters) directives are respectively marked as **d**, **s**, **h**. Names in **`typewriter`** or **boldface** font refer to keywords, while names in ***italics*** refer to variable parameters. **Underlined** parts of keywords refers to shortest abbreviations: Not underlined characters are optional. Expressions between square brackets ([*expression*]) are optional, while alternatives are written in the following way: { *alt_1* | *alt_2* }. To specify angles, lengths, times, energies, atmospheric depths, magnetic fields, etc., it is required to give two fields separated by blank space:

number unit

number is a decimal number and ***unit*** is a character string representing the physical unit used in the specification. All the valid units are listed in table 3.1 (page 38). Additionally, time specifications may be of the form: [***number*** **hr**] [***number*** **min**] [***number*** **sec**], where ***number*** represents a floating point number.

[1] Hidden directives are connected to parameters that seldom need to be modified. They are not printed in the input data summary, unless were explicitly set or a full listing mode was enabled. Notice that this only affects output data printing: All other directive properties remain unchanged.

B.1 List of IDL directives.

#

Comment character. For every scanned input line, all characters placed after the comment character '#' are ignored.

&

Syntax: **&*label***

IDL label. Labels are used by several directives, for example **Remark** and **Skip**. The **&** must be the first non-blank character in the line, and all characters after ***label*** are treated as a comment. ***label*** is a non null string which can contain any character excluding blanks and the comment character **#**.

AddAtmosModel

Syntax: **AddAtmosModel** [***modidstr***] [«] ***input_file***
AddAtmosModel [***modidstr***] [«] **&*label***
. . . *(instructions for model definition)*
&*label*

(d) Adding a custom atmospheric model. ***modidstr*** is a string having no more that 16 characters that uniquely identifies the model being defined. The instructions that define the model can either be read-in from a separate external file ***input_file***, or from a here-document delimited by a label **&*label***. ***modidstr*** can be specified either in the directive line, or within the input data file or here-document. For a more detailed description of the directives to define an atmospheric model, see section 3.3.4 (page 54).

AddSite

Syntax: **AddSite** ***name lat long height***

(d) Appending a new site to the *AIRES site library*. ***name*** is a string having no more than 16 characters, and must be different to all the previously defined sites including the predefined entries listed in table 3.4 (page 57). Site names are case sensitive. ***lat*** and ***long*** are angle specifications defining respectively the geographic latitude and longitude of the site. ***lat*** (***long***) must be in the range $[-90°, 90°]$ ($[-180°, 180°]$). ***height*** is a length specification defining the site's altitude above sea level. The directive **Site** permits to select already defined locations.

AddSpecialParticle

Syntax: **AddSpecialParticle** ***pname module*** [***parstring***]
AddSpecialParticle ***pname module*** [***parstring***] « ***input_file***
AddSpecialParticle ***pname module*** [***parstring***] « ***&label***
. . . (input data for the external module)
&label

(d) Adding a new definition to the list of special particles. ***pname*** is a string having no more than 16 characters that uniquely identifies the special particle being defined. ***module*** is the name of the executable module associated to the special particle. The file ***module*** must exist in the current "working directory" or in one of the directories currently included in the file search path. Every time a new shower with "primary" ***pname*** starts, the module ***module*** will be executed by the main simulation program to generate a list of (standard) primary particles that will be the actual shower primaries. Section 3.5 (page 63) contains a detailed description about how to build and use such kind of modules. ***parstring*** is an optional parameter string (can contain embedded blanks) that is (portably) passed to the external module. Additional input data can be passed to the external module via a ***input_file*** or by a here-document delimited by a label ***&label***.

ADFile

Syntax: **ADFile** [{ **On** | **Off** }]
Default: **ADFile** is equivalent to **ADFile On**
ADFile Off is assumed in case of missing specification.

(d) If **ADFile On** is specified, then an ASCII dump file will be generated upon task completion. The ASCII dump file (ADF) is a portable version of the internal dump file (IDF) that can be transferred among different platforms.

AirAvgZ/A

Syntax: **AirAvgZ/A** ***number***
Default: **AirAvgZ/A 0.5**

(s,h) Sets the value of the average ratio Z/A for air.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

AirRadLength

Syntax: **`AirRadLength`** ***number***
Default: **`AirRadLength 36.62`**

(s,h) Sets the value of the radiation length for air, expressed in $\mathrm{g/cm^2}$.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

AirZeff

Syntax: **`AirZeff`** ***number***
Default: **`AirZeff 7.3`**

(s,h) Sets the value of the effective atomic number Z for air.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

Atmosphere

Syntax: **`Atmosphere`** ***modname*** [***modpars***]
Default: **`Atmosphere Linsley`**

(s) Switches among different atmospheric models. ***modname*** is a string that uniquely identifies the atmospheric model to use. The optional argument(s) represented by ***modpars*** correspond to parameters that some of the supported models may accept. The current version of AIRES includes the following predefined atmospheric models:

1. **Linsley**. Linsley's standard atmosphere model. This is the default model and has no parameters (see section 2.1.2).
2. **SouthPoleAvg**. South Pole average atmosphere. This model has no parameters, and gives a profile obtained from the average of four atmospheric profiles corresponding to typical profiles for the months of March, July, October, and December. This option is recommended for simulations at the South Pole site.
3. **LSouthPole**. Linsley's model for the South Pole. No user-settable model parameters. This model should be used only for simulations with ground level not less than 2000 m.a.s.l.
4. **MalargueAvg**. Malargue site annual average atmosphere.
5. **GAMMA**. GAMMA atmosphere model developed by the La Plata (Argentina) group [41]. Parameter available: **GrdTemp**, temperature at ground.
6. **Isothermic**. Isothermic atmosphere. Parameters available: **Temp**, temperature; **Dens0**, density at sea level.
7. **Homogeneous**. Constant density atmosphere. Parameter available: **Density** (e.g., **`Density 1.22 kg/m3`**); if not specified it defaults to 1.2041 $\mathrm{kg/m^3}$

Not specified ground temperatures default to 295 K.

In addition to the predefined atmospheric models, the user may add custom models using the directive **AddAtmosModel** (107).

Brackets

Syntax: **Brackets** { **On** | **Off** }
Brackets [**On**] ***ob cb*** [***ec***]
Default: **Brackets On { } &**

(d) Controls the behavior of the variable replacement algorithm used while scanning the input file. When the feature is disabled (**Brackets Off**) the input lines are not scanned to search for defined variables to be replaced. When **Brackets On** is in effect and there are defined variables, then variable substitution is performed when it corresponds. The active variable names must be enclosed using the current brackets, which can be changed using this directive. The arguments ***ob***, ***cb***, and ***ec*** correspond, respectively, to the opening and closing brackets, and the bracket escape character. These single character variables must be different, and can be specified with the same rules that apply for the argument of the **CommentCharacter** directive.

CheckOnly

Syntax: **CheckOnly** [{ **On** | **Off** }]
Default: **CheckOnly** is equivalent to **CheckOnly On**
CheckOnly Off is assumed in case of missing specification.

(d) When **CheckOnly** is enabled, the simulation program reads and process all the input data normally, performs the internal consistency checks and then exits without starting the simulations. This directive is useful for input file debugging.

CommentCharacter

Syntax: **CommentCharacter** { ***char*** | ***nnn*** }
Default: The default comment character is '#'

(d) The plain text files produced with the **ExportTables** directive can have heading and trailing lines. All these lines start with a comment character in their first column. The default comment character ('#') is normally OK, but if the **Export**'ed files could be used as input of another program (a plotting utility, for example) which recognizes a different comment character; in such cases the **CommentCharacter** directive permits setting this mentioned character. ***char*** can be any *single* character (with no quotes). Alternatively, the comment character can be specified by means of its ASCII decimal code, expressed in the form of a *three-figure* number ***nnn*** (This permits using non-printable comment characters as well as resetting the comment character to '#').

Date

Syntax: **Date** ***fpyear***
Date ***year month day***
Default: The current date at the moment of invoking the program.

(s) This directive sets the date assumed for the simulations. The date is used at the moment of evaluating the geomagnetic field by means of the IGRF model (see sections 2.1.5 and 3.3.5). Setting the date may be necessary when performing simulations with the purpose of analyzing a certain air shower event reported by an experiment. The date can be specified either as three integers (***year month day***) or a floating point number with the format "year.part_of_the_year".

DelGlobal

Syntax: **DelGlobal** ***var***

(d) Deletes an already defined global variable. See also directives **Import** and **SetGlobal**.

DielectricSuppression

Syntax: **DielectricSuppression** [{ **On** | **Off** }]
Default: **DielectricSuppression** is equivalent to **DielectricSuppression On**
DielectricSuppression On is assumed in case of missing specification.

(s,h) Switch to include/exclude the dielectric suppression effect from the LPM algorithms [25, 31] for the case of electron or positron bremsstrahlung. The effect is **enabled** by default. Disabling it may lead to non realistic air shower simulations. If **LPMEffect Off** is in effect (see page 119), then the dielectric suppression is always disabled.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

DumpFile

Syntax: **DumpFile**
Reserved for future use.

Echo

Syntax: **Echo** ***string***

(d) The action of this directive is to write ***string*** to the standard output channel. Useful to have messages typed while the AIRES input file(s) are being processed. Notice that the message is written *only* to standard output: use the **Remark** (126) directive if it is necessary to save it within the AIRES output files.

ElectronCutEnergy

Syntax: **ElectronCutEnergy** ***energy***
Default: **ElectronCutEnergy 80 KeV**

(s) Minimum kinetic energy for electrons and positrons. Every electron having a kinetic energy below this threshold is not taken into account in the simulation; positrons are forced to annihilation. ***energy*** must be greater than or equal to 80 keV.

ElectronRoughCut

Syntax: **ElectronRoughCut** ***energy***
Default: **ElectronRoughCut 900 KeV**

(s) Electrons and positrons are not followed using detailed calculations when their energy is below the one specified by means of this directive. This means that several processes are not taken into account, for example Coulomb scattering. ***energy*** must be greater than or equal to 45 keV.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

ELimsTables

Syntax: **ELimsTables** ***minenergy maxenergy***
Default: **ELimsTables 10 MeV** ***emax***
emax is the maximum between 10 TeV and $0.75\,E_{\rm primary}$.

(s) This directive defines the energy interval to use in the energy distribution tables (histograms). Each energy distribution histogram consists of 40 logarithmic bins starting with ***minenergy*** (lower energy of bin 1) and ending with ***maxenergy*** (upper energy of bin 40).

EMtoHadronWFRatio

Syntax: **EMtoHadronWFRatio** ***ratio***
Default: **EMtoHadronWFRatio 88**

(s,h) Ratio between the electromagnetic and hadronic thinning weight factors. This instruction permits setting the ratio A_{EM} of equation (2.23). ***ratio*** must be equal or greater than 1. The default value of 88, adjusted taking into account the results of representative simulations, is normally adequate.

End

Syntax: **End**

(d) End of directive stream for the current input file. The file is no more scanned when this directive is found. If **End** is not present, the file is entirely scanned.

Exit

Syntax: **Exit**
x

(d) The program is stopped without taking any further action. This directive is useful to end an interactive session.

ExportPerShower

Syntax: **ExportPerShower** [{ **On** | **Off** }]
Default: **ExportPerShower** is equivalent to **ExportPerShower On**
ExportPerShower Off is assumed in case of missing specification.

(d) This directive affects only those tasks simulated with the **PerShowerData Full** option (see page 122). If **ExportPerShower On** is specified, then a set of plain text files (one file per simulated shower) will be written for all the tables selected for exporting (see directive **ExportTables**). Each one of these "single shower" tables contains the values adopted by the corresponding observable in the respective shower. The normal table containing the average over showers is also exported, and is not affected by this directive.

ExportTables

Syntax: **ExportTables** ***mincode*** [***maxcode***] [**Options** ***optstring***]
ExportTables Clear
Default: No tables are exported by default.

(d) Tables whose codes range from ***mincode*** to ***maxcode*** are selected for exporting as plain text files. If ***maxcode*** is not specified, it is taken equal to ***mincode***. The table codes are integers. A complete list of available tables (more than 180) is placed in appendix B, or can be obtained with directives **Help tables** and/or **TableIndex**. The **Clear** option permits clearing the list of exported tables, thus overriding all the previous **ExportTables** directives. ***opstring*** is a string of characters to set available options: **s** (**h**) suppress (include) file header; **x** (**X**) include "border" bins as comments (within the data); **U** do not include "border" bins; **r** (**d**) normal (density) lateral distributions; **L** (**l**) distributions normalized as $d/d\log_{10}$ ($d/d\ln$); **r** (**a**) express atmospheric depth as vertical (slant) depths; **K**, **M**, **G**, **T**, **P**, **E**, express energies in keV, MeV, ..., EeV. The default options are: **hxrG**.

ExtCollModel

Syntax: **ExtCollModel** [{ **On** | **Off** }]
Default: **ExtCollModel** is equivalent to **ExtCollModel On**
ExtCollModel On is assumed in case of missing specification.

(s) Switch to enable/disable the external hadronic interactions model.

This directive belongs to the model-dependent IDL instruction set and may be changed or not

implemented in future versions of AIRES.

ExtNucNucMFP

Syntax: **ExtNucNucMFP** [{ **On** | **Off** }]
Default: **ExtNucNucMFP** is equivalent to **ExtNucNucMFP On**
ExtNucNucMFP On is assumed in case of missing specification.

(s,h) Switch to enable/disable calculation of mean free paths for nucleus-nucleus collisions via the corresponding external hadronic interactions model.

If the switch is set to **Off** then the nucleus-nucleus mean free paths are evaluated using an AIRES built-in procedure. In this case the mean free paths are obtained by scaling properly the corresponding proton-nucleus mean free path.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

FileDirectory

Syntax: **FileDirectory** ***dopt directory***
Default: The output and scratch directories default to the current (working) directory. The global and export directories default to the current value of the output directory.

(d) This directive sets the output file directories. ***dopt*** is a character string that can take any one of the following values: **All**, **Output**, **Global**, **Export**, or **Scratch**. These alternatives permit setting all the AIRES directories defined in section 3.3.2 (page 50). The option **All** can be used to simultaneously set the "output" (compressed file), "global" and "export" directories. ***directory*** is a character string not longer than 94 characters that must be recognized by the operating system as a valid directory.

FirstShowerNumber

Syntax: **FirstShowerNumber** ***fshowerno***
Default: **FirstShowerNumber 1**

(s) A positive integer in the range $[1, 759375]$ indicating the number to be assigned to the first simulated shower. The shower number is used in tables 5000 to 5513, and in the "beginning of shower" and "end of shower" compressed file records (for details see chapter 4).

ForceInit

Syntax: **ForceInit** [{ **On** | **Off** }]
Default: **ForceInit** is equivalent to **ForceInit On**
ForceInit Off is assumed in case of missing specification.

(d) If **ForceInit** is enabled, then a new task is started at the beginning of every process. If the corresponding IDF file exists, then the task version is increased until an unused version is found. This directive is useful for debugging purposes.

ForceLowEAnnihilation

Syntax: **ForceLowEAnnihilation** ***opt***

Default: **ForceLowEAnnihilation Normal**
ForceLowEAnnihilation with no specification is equivalent to **ForceLowEAnnihilation Always**.

(s,h) Directive to control the action to take when processing a low energy particle that can annihilate with its respective anti-particle. The variable ***opt*** can take the values **Always**, **Never**, or **Normal**. The first two alternatives correspond, respectively, to the cases where the low energy particles will always be forced to annihilation or be discarded without producing any secondary particle. In the (default) **Normal** option the action to take for annihilating low energy particles depends on the particle cut energy and mass: If the cut energy is less (greater) than the rest mass then the particle is (is not) forced to annihilation.

ForceLowEDecays

Syntax: **ForceLowEDecays** ***opt***

Default: **ForceLowEDecays Normal**
ForceLowEDecays with no specification is equivalent to **ForceLowEDecays Always**.

(s,h) Directive to control the action to take when processing a low energy unstable particle that can decay into other particles. The variable ***opt*** can take the values **Always**, **Never**, or **Normal**. The first two alternatives correspond, respectively, to the cases where the low energy particles will always be forced to decays or be dicarded without producing any secondary particle. In the (default) **Normal** option the action to take for decaying low energy particles depends on the particle cut energy and mass: If the cut energy is less (greater) than the rest mass then the particle is (is not) forced to decays.

ForceModelName

Syntax: **ForceModelName** ***modsel***

Default: No model name check is performed when this directive is not used.

(s) This directive allows the user to force that a given input data set will be processed with the simulation program linked with the external collision package specified with ***modsel***. Currently ***modsel*** can be one of (case dependent!) EPOS-LHC3400, EPOS1990, QGSJET-II-04, QGSJET-II-03, SIBYLL23c, SIBYLL231, or SIBYLL21. This directive is useful as a security tool to allow execution of simulations only if the executable being used is the one that corresponds to the selected hadronic model. Consider also using it together with **StopOnError**.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

GammaCutEnergy

Syntax: **GammaCutEnergy** ***energy***
Default: **GammaCutEnergy 80 KeV**

(s) Minimum energy for gammas. Every gamma ray having an energy below this threshold is not taken into account in the simulation. ***energy*** must be greater than or equal to 80 keV.

GammaRoughCut

Syntax: **GammaRoughCut** ***energy***
Default: **GammaRoughCut 750 KeV**

(s) Gamma rays are not followed using detailed calculations when their energy is below the one specified by means of this directive. This means that several processes are not taken into account, for example pair production. ***energy*** must be greater than or equal to 45 keV.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

GeomagneticField

Syntax: **GeomagneticField** [{ **On** | **Off** }]
GeomagneticField ***stg*** [***inc*** [***dec***]] [**Fluctuations** ***fluc***]
GeomagneticField [**On**] **Fluctuations** ***fluc***
Default: **GeomagneticField Off** when there is no **Site** specification;
GeomagneticField On otherwise.

(s) Setting the geomagnetic field manually and/or enabling magnetic fluctuations. ***stg*** must be a valid magnetic field strength specification, and ***inc*** and ***dec*** are angle specifications. Such fields correspond respectively to the geomagnetic field strength, F, and to the inclination, I, and declination, D, angles defined in section 2.1.5 (page 17). When one or more of such parameters are entered by means of the **GeomagneticField** directive, they override the respective values that are calculated automatically using the IGRF model [15], as explained in section 3.3.5 (page 56). The fluctuation specification ***fluc*** adopts three different formats: (i) *Absolute:* In this case ***fluc*** represents a (positive) magnetic field strength. (ii) *Relative:* ***fluc*** adopts the format ***number*** **Relative**, and refers to the ratio between the actual fluctuation strength and the average value of the magnetic field. (iii) *In percent:* ***fluc*** adopts the format ***number*** **%**. ***number*** corresponds to a relative specification multiplied by 100. The effect of magnetic field fluctuations is explained in section 3.3.5 (page 56).

GroundAltitude

Syntax: **GroundAltitude** ***altdepth***
GroundDepth ***altdepth***
Default: The altitude of the site currently in effect.

(s) Ground level altitude. ***altdepth*** can be either a length specification (ranging from 0 to 112 km) or an atmospheric depth specification (ranging from 0 to 1033 g/cm^2).

Help

Syntax: **Help** [{ * | **tables** | **sites** }]
help [{ * | **tables** | **sites** }]
? [{ * | **tables** | **sites** }]

(d) The action of the **Help** directive is to type a brief summary of IDL directives, output data tables (histograms) or sites defined in the AIRES site library. **Help *** gives a full IDL directive list, including all "hidden" directives. The **?** form is equivalent to the combined action of **Help** and **Prompt On**

Import

Syntax: **Import** [{ **Dynamic** | **Static** }] ***varname***
Default: No environmental variables are imported by default.

(d) Importing environment variables. The operating system environment variable ***varname*** is imported and stored as an active variable that can either be used within the IDL input stream or passed to the compressed output files or special primary modules. The **Dynamic** qualifier (default) indicates the *dynamic* character of the corresponding variable. This means that the value currently passed to the external modules is modified each time AIRES is invoked for a given task. On the other hand, **Static** variables are set at the first invocation of AIRES; and further settings have no effect.

ImportShell

Syntax: **ImportShell** [{ **Dynamic** | **Static** }] ***pname shell_instruction***
Default: Nothing imported by default.

(d) Importing the output of shell commands. The data written to standard output when the operating system executes the instruction ***shell_instruction*** is imported and stored as the active variable ***pname*** that can either be used within the IDL input stream or passed to the compressed output files or special primary modules. The **Dynamic** qualifier (default) indicates the *dynamic* character of the corresponding variable. This means that the value currently passed to the external modules can be modified each time AIRES is invoked for a given task. On the other hand, **Static** variables are set at the first invocation of AIRES; and further settings have no effect.

InjectionAltitude

Syntax: **InjectionAltitude** ***altdepth***
InjectionDepth ***altdepth***
Default: **InjectionAltitude 100 km**

(s) Primary injection altitude. ***altdepth*** can be either a length specification (ranging from 0 to 112 km) or an atmospheric depth specification (ranging from 0 to 1033 g/cm^2).

Input

Syntax: **Input** ***file*** [***arg1*** [***arg2*** [**...**]]]

(d) File ***file*** is inserted in the input data stream. **Input** directives can be nested. The search path for locating input files include the "working directory" (see section 3.3.2), the directories that are included by default, and all the directories that were specified with directive **InputPath**. Optionally, it is possible to pass arguments to the included file. They are assigned to global variables named "**1**", "**2**", ..., and will be visible for the directives placed within **file**.

InputListing

Syntax: **InputListing** [{ **Brief** | **Full** }]
Default: **InputListing** is equivalent to **InputListing Brief**
InputListing Brief is assumed in case of missing specification.

(d) Data related to hidden input directives are not printed in the output summary file unless the corresponding variables were explicitly set or **InputListing Full** was specified.

InputPath

Syntax: **InputPath** [{ **Insert** | **Append** }] [***dir1***[:***dir2***[: . . .]]]
Default: The file search path contains by default the "working directory", and the directories coming with the AIRES distribution that contain input files and/or executable modules that could be invoked from within a set of IDL directives.

(d) Modifying the directory search path for the files included with the **Input** directive and/or other directives that request external files. This directive can be used multiple times if required. Different search directories can be specified in a single invocation separating them with colons (:) with no embedded blanks. The keyword **Append** indicates that the specified directory(ies) must be appended to the ones already inserted, while **Insert** indicates that such directory(ies) must be inserted at the beginning of the current path string. If **InputPath** is invoked with no arguments, then the search path is cleared.

LaTeX

Syntax: **LaTeX** [{ **On** | **Off** }]
Default: **LaTeX** is equivalent to **LaTeX On**
LaTeX Off is assumed in case of missing specification.

(d) If **LaTeX On** is specified, then the output summary file is written using the LaTeX word processor format. Otherwise it is written as a plain text file. When this option is enabled, a TeX file ***taskname*.tex** is created simultaneously with the summary file.

LPMEffect

Syntax: **LPMEffect** [{ **On** | **Off** }]
Default: **LPMEffect** is equivalent to **LPMEffect On**
LPMEffect On is assumed in case of missing specification.

(s,h) Switch to include/exclude the Landau-Pomeranchuk-Migdal effect [30, 25] from the electron-positron and gamma propagating algorithms. The effect is **enabled** by default. Disabling it may lead to non realistic air shower simulations. If **LPMEffect Off** is in effect, then the dielectric suppression is also disabled (see page 111).

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

MaxCpuTimePerRun

Syntax: **MaxCpuTimePerRun** { ***time*** | **Infinite** }
Default: **MaxCpuTimePerRun Infinite**

(d) This directive sets the maximum CPU time for individual runs, being a *run* the processing chunk that goes between two consecutive updates of the internal dump file. This parameter does not impose any restriction on the CPU time available for the simulation of a single shower (or a group of them), which is always infinite. ***time*** is any valid time specification. See also directives **RunsPerProcess** and **ShowersPerRun**.

MesonCutEnergy

Syntax: **MesonCutEnergy** ***energy***
Default: **MesonCutEnergy 60 MeV**

(s) Minimum kinetic energy for mesons (pions, kaons, etc.). Every meson having a kinetic energy below this threshold is not taken into account in the simulation; unstable particles are forced to decays. ***energy*** must be greater than or equal to 500 keV.

MFPHadronic

Syntax: **MFPHadronic** ***mfpsel***

(s) Directive to select among different sets of mean free paths parameterizations. ***mfpsel*** is a

character string that identifies the set to be used (see page 61).

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

MFPThreshold

Syntax: **MFPThreshold** ***energy***
Default: **MFPThreshold 50 GeV**

(s,h) Threshold energy for the currently effective mean free paths. All hadronic collisions with energy greater than or equal to this threshold will be processed using the current mfp parameterization (that can be set using directive **MFPHadronic**); otherwise standard MFP's will be used. ***energy*** must be greater than or equal to 200 MeV.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

MinExtCollEnergy

Syntax: **MinExtCollEnergy** ***energy***
Default: **MinExtCollEnergy 200 GeV** for the SIBYLL model;
MinExtCollEnergy 80 GeV for the QGSJET model.

(s,h) Threshold energy for invoking the external hadronic collision routine (if enabled). ***energy*** must be greater than or equal to 25 GeV.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

MinExtNucCollEnergy

Syntax: **MinExtNucCollEnergy** ***energypernucleon***
Default: **MinExtCollEnergy 200 GeV** for the SIBYLL model;
MinExtCollEnergy 80 GeV for the QGSJET model.

(s,h) Threshold energy per nucleon for invoking the external nucleus-nucleus collision routine (if enabled). ***energypernucleon*** must be greater than or equal to 25 GeV.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

MuonBremsstrahlung

Syntax: **Muonbremsstrahlung** [{ **On** | **Off** }]
Default: **Muonbremsstrahlung** is equivalent to **Muonbremsstrahlung On**
Muonbremsstrahlung On is assumed in case of missing specification.

(s,h) Switch to include/exclude the muon bremsstrahlung [24] and muonic pair production processes from the muon propagating algorithms. These interactions are **enabled** by default.

Disabling them may lead to non realistic air shower simulations.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

MuonCutEnergy

Syntax: **MuonCutEnergy** ***energy***
Default: **MuonCutEnergy 10 MeV**

(s) Minimum kinetic energy for muons. Every muon having a kinetic energy below this threshold is not taken into account in the simulation; it is forced to a decay. ***energy*** must be greater than or equal to 500 keV.

NuclCollisions

Syntax: **NuclCollisions** [{ **On** | **Off** }]
Default: **NuclCollisions** is equivalent to **NuclCollisions On**
NuclCollisions On is assumed in case of missing specification.

(s,h) Switch to include/exclude the hadronic inelastic collisions with air nucleus from the heavy particles propagating algorithms. The collisions are **enabled** by default. Disabling them may lead to non realistic air shower simulations.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

NuclCutEnergy

Syntax: **NuclCutEnergy** ***energy***
Default: **NuclCutEnergy 120 MeV**

(s) Minimum kinetic energy for nucleons and nuclei. Every such particle having a kinetic energy below this threshold is not taken into account in the simulation. ***energy*** must be greater than or equal to 500 keV.

ObservingLevels

Syntax: **ObservingLevels** ***nofol*** [***altdepth1*** ***altdepth2***]
Default: **ObservingLevels 19**

(s) This directive defines the number and position of the observing levels used for longitudinal development recording (see page 23) . ***altdepth1*** and ***altdepth2*** are altitude (or atmospheric depth) specifications that define the positions of the first and last observing levels. ***nofol*** is an integer that sets the number of observing levels. It must lie in the range $[4, 510]$. The observing levels are equally spaced in atmospheric depth units. The first (last) level corresponds to the highest (lowest) altitude.

If ***altdepth1*** and ***altdepth2*** are not specified, then the observing levels are placed between the injection and ground planes, but spacing them differently (see section 3.3.3): The injection level corresponds to observing level "0" while the ground level corresponds to observing level "***nofol*** + 1". For example, if the injection (ground) level is placed at 0 (1000) g/cm^2, the directive **ObservingLevels 19** will set 19 observing levels placed at depths 50, 100, 150, ..., 950 g/cm^2.

OutputListing

Syntax: **OutputListing** [{ **Brief** | **Full** }]
Default: **OutputListing** is equivalent to **OutputListing Brief**
OutputListing Brief is assumed in case of missing specification.

(d) Hidden output data items are not printed in the output summary file unless **OutputListing Full** is specified.

PerShowerData

Syntax: **PerShowerData** ***option***
Default: **PerShowerData** is equivalent to **PerShowerData Full**
PerShowerData Brief is assumed in case of missing specification.

(s) Directive to control the amount of individual shower data to be stored after each shower is completed. ***option*** is a character string that can take any one of the following values: **None**, **Brief** or **Full**. When **None** is specified, no individual shower data is saved. The **Brief** level implies saving global parameters such as the depth of shower maximum $X_{\max}$, for example; and the **Full** level is the **Brief** level plus all the single shower tables (see page 113).

PhotoNuclear

Syntax: **PhotoNuclear** [{ **On** | **Off** }]
Default: **PhotoNuclear** is equivalent to **PhotoNuclear On**
PhotoNuclear On is assumed in case of missing specification.

(s,h) Switch to include/exclude the inelastic collisions gamma-air nucleus (photonuclear reactions) from the gamma ray propagating algorithms. The collisions are **enabled** by default. Disabling them may lead to non realistic air shower simulations.

This directive belongs to the model-dependent IDL instruction set and may be changed or not implemented in future versions of AIRES.

PrimaryAzimAngle

Syntax: **PrimaryAzimAngle** ***minang*** [***maxang***] [{ **Magnetic** | **Geographic** }]
Default: **PrimaryAzimAngle 0 deg Magnetic** if the zenith angle is fixed;
PrimaryAzimAngle 0 deg 360 deg Magnetic otherwise
(see **PrimaryZenAngle**).

(s) Primary azimuth angle. The angle for each shower is selected with uniform probability distribution in the interval [***minang***, ***maxang***]. If the angle ***maxang*** is not specified, it is taken equal to ***minang*** (fixed azimuth angle). The **Geographic** keyword indicates that the specified azimuth is measured with respect to the *geographic* north, positive for eastwards directions; in this case the azimuth angle used by AIRES is obtained applying equation (3.8). If no keyword or the **Magnetic** keyword is specified, then the origin for the azimuths is the *magnetic* north, and the given angles are interpreted accordingly with the orientation of the AIRES coordinate system defined in section 2.1.1 (page 9).

PrimaryEnergy

Syntax: **PrimaryEnergy** ***minener*** [***maxener*** [***gamma***]]
Default: None.

(s) Energy of primary. If only ***minener*** is specified then all primaries have a fixed energy equal to this parameter. Otherwise the energy will be sampled from the interval $[E_{\rm min}, E_{\rm max}] = [$***minener***, ***maxener***$]$ with the probability distribution of equation (3.2) with exponent γ optionally specified by ***gamma***.

The primary energy must be larger than 500 MeV and less than 3×10^{12} GeV (3×10^{21} eV). There are no restrictions on γ. If not specified it is set to 1.7.

PrimaryParticle

Syntax: **PrimaryParticle** ***particle*** [***weight***]
Default: None. This directive is always required.

(s) Primary particle specification. ***particle*** is the particle name. **Proton**, **Iron**, **Feˆ56**, etc. are valid particle names. Special particle names defined by means of directive **AddSpecialParticle** can also be used with this instruction. If more than one **PrimaryParticle** directive appear within the input instructions, then the primary particles will be selected at random among the different specified particle kinds, with probabilities proportional to the weights specified in the corresponding ***weight*** fields. If ***weight*** is not specified, then the particle weight is taken as 1.

PrimaryZenAngle

Syntax: **PrimaryZenAngle** ***minang*** [***maxang*** [{ **S** | **SC** | **CS** }]]
Default: **PrimaryZenAngle 0 deg**

(s) Primary zenith angle, Θ. If only ***minang*** is specified, then the zenith angle is fixed and equal to this value, and the default for the azimuth angle will be 0. Otherwise the zenith angle for each shower is selected randomly within the interval $[$***minang***, ***maxang***$]$, with the *sine* probability distribution of equation (3.4), which is proportional to $\sin\Theta$ (default or **S** specification), or the *sine-cosine* probability distribution of equation (3.6), which is proportional to $\sin\Theta\cos\Theta$ (**SC** or **CS** specifications). In this case the default for the azimuth angle is **PrimaryAzimAngle 0 deg 360 deg**. Both ***minang*** and ***maxang*** must belong to the interval $[0°, 90°)$.

PrintTables

Syntax: **PrintTables** ***mincode*** [***maxcode***] [**Options** ***optstring***]
PrintTables Clear
Default: No tables are printed by default.

(d) Tables whose codes range from ***mincode*** to ***maxcode*** are selected for being displayed in the summary output file. If ***maxcode*** is not specified, it is taken equal to ***mincode***. The table codes are integers. A complete list of available tables (more than 180) is placed in appendix C (page 132), or can be obtained with directives **Help tables** and/or **TableIndex**. The **Clear** option permits clearing the list of printed tables, thus overriding all the previous **PrintTables** directives. ***opstring*** is a string of characters to set available options: **n** suppress plotting minimum (**<**) and maximum (**>**) characters; **m** include minimum and maximum plots in the tables; **M** do not insert character plots, make a completely numerical table instead; **S** (**R**) use standard deviations (RMS errors of the means) to plot error bars; **r** (**d**) normal (density) lateral distributions; **L** (**l**) distributions normalized as $d/d\log_{10}$ ($d/d\ln$). The default options are: **nSr**.

Prompt

Syntax: **Prompt** [{ **On** | **Off** }]
Default: **Prompt** is equivalent to **Prompt On**
Prompt Off is assumed in case of missing specification.

(d) Turns prompting on/off. This directive is meaningful only in interactive sessions.

PropagatePrimary

Syntax: **PropagatePrimary** [{ **On** | **Off** }]
Default: **PropagatePrimary** is equivalent to **PropagatePrimary On**
PropagatePrimary On is assumed in case of missing specification.

(s,h) This directive controls the initial propagation of the primary. If the **On** option is selected (the default), then the primary is normally advanced before the first interaction takes place, and

therefore the first interaction altitude will be variable. Otherwise the first interaction will be forced to occur at the injection altitude.

This directive is ignored for showers initiated by special primaries (see section 3.5).

RandomSeed

Syntax: **RandomSeed** ***seed***
RandomSeed GetFrom ***idfile***
Default: **RandomSeed 0.0**

(s) This directive sets the random number generator seed. ***seed*** is a real number. If it belongs to the interval $(0, 1)$ then the seed is effectively taken as the given number. Otherwise it is evaluated internally (using the system clock). The alternative syntax with the keyword **GetFrom** allows extracting the random generator seed from an already existing internal dump file. This is most useful to reproducing a previous simulation repeating the original random number simulator configuration.

RecordObsLevels

Syntax: **RecordObsLevels** [**Not**] [***lev1*** [***lev2*** [***step***]]]
RecordObsLevels [**Not**] { **All** | **All**/***step*** | **None** }
Default: **RecordObsLevels All**

(s) Directive to mark a certain subset of the defined observing levels for inclusion (or exclusion) in the set of levels that are included in the longitudinal tracking compressed particle file. The integer variables ***lev1 lev2*** and ***step*** are the arguments of a FORTRAN do loop which starts at ***lev1***, ends at ***lev2*** advancing in steps of ***step***. The keyword **Not** indicates that the corresponding levels must be *excluded* for being recorded in the file. If ***lev2*** and/or ***step*** are not indicated they default to ***lev1*** and 1 respectively. **RecordObsLevels All/*step*** is a short form for **RecordObsLevels** 1 N_o ***step***, where N_o is the number of defined observing levels. **RecordObsLevels All** is equivalent to **RecordObsLevels All/1** while **RecordObsLevels None** can be used in place of **RecordObsLevels Not All**. This directive can be repeatedly used within an input instruction stream to mark or unmark arbitrary subsets of observing levels, as explained in page 86.

RecordSpecPrimaries

Syntax: **RecordSpecPrimaries** [{ **On** | **Off** }]
Default: **RecordSpecPrimaries** is equivalent to **RecordSpecPrimaries On**
RecordSpecPrimaries On is assumed in case of missing specification.

(s) Directive to enable or disable recording in the compressed output files the list of special primary particles injected at the beginning of each shower.

Remark

Syntax: **Remark** ***string***
Remark ***&label***
First line of remarks.
...
Last line of remarks.
&label

(s) Remarks directive. Each time this directive appears in the input data stream, the corresponding remark string(s) are appended to the remarks text. All the entered remarks will be printed in the log and summary files, and stored in different output data files. There is no limit in the number of remark lines, but every line cannot be longer than 75 characters. Notice that the **Echo** (111) directive can also be used if what is needed is to type a message to standard output while the AIRES input file(s) are being processed.

ResamplingRatio

Syntax: **ResamplingRatio** ***rsratio***
Default: **ResamplingRatio 10**

(s) This directive sets the variable s_r used in the resampling algorithm defined in section 4.2.1 (page 82). ***rsratio*** is a real number that must be greater or equal than 1.

RLimsFile

Syntax: **RLimsFile** ***filext rmin rmax***
Default: **RLimsFile** ***any_file*** **250 m 12 km**

(s) This directive defines the lateral limits for the compressed data file whose extension is ***filext***. For the ground particle file, ***rmin*** and ***rmax*** define, together with the resampling ratio that is controlled by the IDL instruction **ResamplingRatio** the radial limits of the zone where the particles are going to be saved (see page 82). In the case of longitudinal tracking particle files, those parameters define the inclusion zone at ground level. At an arbitrary altitude, the particles are included accordingly with the rules explained in section 4.2.1 (page 76).

RLimsTables

Syntax: **RLimsTables** ***rmin rmax***
Default: **RLimsTables 50 m 2 km**

(s) This directive defines the radial interval to use in the lateral distribution tables (histograms). Each lateral distribution histogram consists of 40 logarithmic bins starting with ***rmin*** (lower radius of bin 1) and ending with ***rmax*** (upper radius of bin 40).

RunsPerProcess

Syntax: **RunsPerProcess** { ***number*** | **Infinite** }
Default: **RunsPerProcess Infinite**

(d) Number of runs within a process (see also **MaxCpuTimePerRun** and **ShowersPerRun**).

SaveInFile

Syntax: **SaveInFile** ***filext particle1*** [***particle2***] ...
Default: **SaveInFile grdpcles All**
SaveInFile lgtpcles None

(s) This directive allows to control the particles being saved in the compressed file whose extension is ***filext*** (see directive **RLimsFile**). ***particle1***, ***particle2***, ..., are valid particle or particle group names. This directive, together with **SaveNotInFile** are useful to save output file space in certain circumstances.

SaveNotInFile

Syntax: **SaveNotInFile** ***filext particle1*** [***particle2***] ...

(s) The syntax of this directive is similar to **SaveInFile**, and its meaning is opposite (**SaveInFile** ***filext*** **None** is equivalent to **SaveNotInFile** ***filext*** **All**).

SeparateShowers

Syntax: **SeparateShowers** { **Off** | ***number*** }
Default: **SeparateShowers Off**

(s) In a task involving more than one shower, the compressed output files can be split into several pieces each one storing the data corresponding to ***number*** showers. In particular, **SeparateShowers 1** generates one compressed file per shower while **SeparateShowers Off** disables file splitting.

SetGlobal

Syntax: **SetGlobal** [{ **Dynamic** | **Static** }] ***varname value***
Default: No environmental variables are imported by default.

(d) Setting global variables. The variable ***varname*** is set to the string ***value***. If the variable was already set, then its old setting is superseded. The defined variables can either be used within the IDL input stream or passed to the compressed output files or special primary modules. The **Dynamic** qualifier (default) indicates the *dynamic* character of the corresponding variable. This means that the value currently passed to the external modules is modified each time AIRES is invoked for a given task. On the other hand, **Static** variables are set at the first invocation of AIRES; and further settings have no effect.

SetTimeAtInjection

Syntax: **SetTimeAtInjection** [{ **On** | **Off** }]
Default: **SetTimeAtInjection** is equivalent to **SetTimeAtInjection On**
SetTimeAtInjection On is assumed in case of missing specification.

(s,h) Directive to set whether the time count for each shower is started at the moment of injecting the primary particle (**On**) or at its first interaction (**Off**).

This directive is ignored for showers initiated by special primaries (see section 3.5).

SetTopAtInjection

Syntax: **SetTopAtInjection** [{ **On** | **Off** }]
Default: **SetTopAtInjection** is equivalent to **SetTopAtInjection On**
SetTopAtInjection On is assumed in case of missing specification.

(d) When this switch is enabled, the top surface of the shower bounding box is set accordingly with the atmospheric depth of the primary injection point. Otherwise it is set accordingly with the depth of the first primary interaction.

Shell

Syntax: **Shell** ***shell_instruction***
Shell ***shell_instruction*** **«** ***input_file***
Shell ***shell_instruction*** **«** **&*label***
. . . (lines with input data)
&*label*

(d) Executing a given shell instruction. The string ***shell_instruction*** is passed to the operating system to have it executed within the current shell interpreter. It is optionally possible to specify a input data file or, alternatively, place such input data as a here-document delimited by a label **&*label***.

ShowersPerRun

Syntax: **ShowersPerRun** { ***number*** | **Infinite** }
Default: **ShowersPerRun Infinite**

(d) Maximum number of showers in a run (see also **MaxCpuTimePerRun** and **RunsPerProcess**). Notice that this parameter is related with the computer environment only and does not affect the total number of showers that define a task (see **TotalShowers**).

Site

Syntax: **Site** ***name***
Default: **Site Site00**

(s) The **Site** directive specify the geographical location that define the environment (latitude, longitude and altitude) where the simulations take place. ***name*** is a string identifying the selected site. It must either be one of the predefined sites of the AIRES site library, listed in table 3.4 (page 57), or have been previously defined by means of the **AddSite** directive.

Skip

Syntax: **Skip** ***&label***

(d) Instruction to skip part of an input data stream. All directives placed after the **Skip** statement and before ***&label*** are skipped. Notice that this is not a "go to" statement: It is only possible to skip forwards, never backwards.

SpecialParticLog

Syntax: **SpecialParticLog** ***lvl***
Default: **SpecialParticLog** is equivalent to **SpecialParticLog 1**
SpecialParticLog 0 is assumed in case of missing specification.

(d) Controlling the amount of data related with special primary particles to be saved in the corresponding log file. ***lvl*** is an integer parameter that can take the following values:

0 No information written in the log file.
1 Messages before and after invoking the external module.
2 Level 1 plus detailed list of valid primaries.

SPMaxFieldsToAdd

Syntax: **SPMaxFieldsToAdd** ***mxdynfields***
Default: **SPMaxFieldsToAdd 0**

(s) Sets the maximum number of fields that can be dynamically added to the AIRES compressed output files.

StackInformation

Syntax: **StackInformation** [{ **On** | **Off** }]
Default: **StackInformation** is equivalent to **StackInformation On**
StackInformation Off is assumed in case of missing specification.

(d) Directive to instruct AIRES to print detailed stack usage information in the summary output file.

StopOnError

Syntax: **StopOnError** [{ **On** | **Off** }]
Default: **StopOnError** is equivalent to **StopOnError On**
StopOnError Off is assumed in case of missing specification.

(d) Directive to enable or disable the ***"stop-on-error"*** condition. When this condition is in effect, the severity of every error that could happen while parsing the AIRES IDL directives is set to the maximum level (fatal error). In consequence, the simulation program will always abort if there are errors within the input instructions. This directive is useful as a security tool to allow execution of simulations only if the input files do not contain instructions with errors. Consider also using it together with **ForceModel**.

Summary

Syntax: **Summary** [{ **On** | **Off** }]
Default: **Summary** is equivalent to **Summary On**
Summary On is assumed in case of missing specification.

(d) Directive to enable or disable the output summary.

TableIndex

Syntax: **TableIndex** [{ **On** | **Off** }]
Default: **TableIndex** is equivalent to **TableIndex On**
TableIndex Off is assumed in case of missing specification.

(d) Directive to instruct AIRES to print a table index in the summary output file.

TaskName

Syntax: **TaskName** [**Append**] ***taskname*** [***taskversion***]
Default: **TaskName GIVE_ME_A_NAME_PLEASE**

(d) Task name assignment. ***taskname*** is a character string which identifies the current task. If its length is greater than 64 characters, it will be truncated to the first 64 characters. ***taskversion*** is an optional integer between 0 (default) and 999. If ***taskversion*** is not zero, the effective task name is ***taskname_taskversion***. If the keyword **Append** is used, then ***taskname*** is appended to the existing task name string. The task name is used to set the file names of all output files.

ThinningEnergy

Syntax: **ThinningEnergy** { ***energy*** | ***number*** **Relative** }
Default: **ThinningEnergy 1.0e-4 Relative**

(s) Thinning energy. It can be expressed either as an absolute energy or as a real (positive) number with the keyword **Relative** (In this case the thinning energy is the primary energy times the specified number).

ThinningWFactor

Syntax: **ThinningWFactor** ***number***
Default: **ThinningWFactor 12**

(s,h) Thinning weight factor. This instruction permits setting the weight factor W_f of equation (2.23).

TotalShowers

Syntax: **TotalShowers** ***nofshowers***
Default: None. This directive is always required.

(d) Total number of showers. ***nofshowers*** is a positive integer in the range $[1, 759375]$ defining the number of showers to be simulated in the current task. Notice that this is a dynamic parameter, that is, it can be modified (either enlarged or reduced) during the simulations.

Trace

Syntax: **Trace** [{ **On** | **Off** }]
Default: **Trace** is equivalent to **Trace On**
Trace Off is assumed in case of missing specification.

(d) Directive to enable or disable input data tracing. If enabled (**On**) then trace information about the directives being processed by the IDL parser is written into the standard output channel. This directive is useful to debug IDL input data sets.

TSSFile

Syntax: **TSSFile** [{ **On** | **Off** }]
Default: **TSSFile** is equivalent to **TSSFile On**
TSSFile Off is assumed in case of missing specification.

(d) If **TSSFile On** is specified, then a task summary script file will be generated upon task completion. The task summary script file (TSS) is a plain text file containing information about the main parameters of the simulation, in the format **Keyword** = ***value***, suitable for processing with other programs.

Appendix C

Output data table index

We list here all the tables defined in AIRES 19.04.08. These tables can be processed using directives **PrintTables** and/or **ExportTables** (see chapter 3).

	Code	*Table name*
1	**0100**	Atmospheric profile.
2	**1001**	Longitudinal development: Gamma rays.
3	**1005**	Longitudinal development: Electrons.
4	**1006**	Longitudinal development: Positrons.
5	**1007**	Longitudinal development: Muons (+).
6	**1008**	Longitudinal development: Muons (−).
7	**1011**	Longitudinal development: Pions (+).
8	**1012**	Longitudinal development: Pions (−).
9	**1013**	Longitudinal development: Kaons (+).
10	**1014**	Longitudinal development: Kaons (−).
11	**1021**	Longitudinal development: Neutrons.
12	**1022**	Longitudinal development: Protons.
13	**1023**	Longitudinal development: Antiprotons.
14	**1041**	Longitudinal development: Nuclei.
15	**1091**	Longitudinal development: Other charged pcles.
16	**1092**	Longitudinal development: Other neutral pcles.
17	**1205**	Longitudinal development: e+ and e−
18	**1207**	Longitudinal development: mu+ and mu−
19	**1211**	Longitudinal development: pi+ and pi−
20	**1213**	Longitudinal development: K+ and K−
21	**1291**	Longitudinal development: All charged particles.
22	**1292**	Longitudinal development: All neutral particles.
23	**1293**	Longitudinal development: All particles.

	Code	*Table name*
24	**1301**	Unweighted longit. development: Gamma rays.
25	**1305**	Unweighted longit. development: Electrons.
26	**1306**	Unweighted longit. development: Positrons.
27	**1307**	Unweighted longit. development: Muons (+).
28	**1308**	Unweighted longit. development: Muons (−).
29	**1311**	Unweighted longit. development: Pions (+).
30	**1312**	Unweighted longit. development: Pions (−).
31	**1313**	Unweighted longit. development: Kaons (+).
32	**1314**	Unweighted longit. development: Kaons (−).
33	**1321**	Unweighted longit. development: Neutrons.
34	**1322**	Unweighted longit. development: Protons.
35	**1323**	Unweighted longit. development: Antiprotons.
36	**1341**	Unweighted longit. development: Nuclei.
37	**1391**	Unweighted longit. development: Other charged pcles.
38	**1392**	Unweighted longit. development: Other neutral pcles.
39	**1405**	Unweighted longit. development: e+ and e−
40	**1407**	Unweighted longit. development: mu+ and mu−
41	**1411**	Unweighted longit. development: pi+ and pi−
42	**1413**	Unweighted longit. development: K+ and K−
43	**1491**	Unweighted longit. development: All charged particles.
44	**1492**	Unweighted longit. development: All neutral particles.
45	**1493**	Unweighted longit. development: All particles.
46	**1501**	Longitudinal development: Energy of gamma rays.
47	**1505**	Longitudinal development: Energy of electrons.
48	**1506**	Longitudinal development: Energy of positrons.
49	**1507**	Longitudinal development: Energy of muons (+).
50	**1508**	Longitudinal development: Energy of muons (−).
51	**1511**	Longitudinal development: Energy of pions (+).
52	**1512**	Longitudinal development: Energy of pions (−).
53	**1513**	Longitudinal development: Energy of kaons (+).
54	**1514**	Longitudinal development: Energy of kaons (−).
55	**1521**	Longitudinal development: Energy of neutrons.
56	**1522**	Longitudinal development: Energy of protons.
57	**1523**	Longitudinal development: Energy of antiprotons.
58	**1541**	Longitudinal development: Energy of nuclei.
59	**1591**	Longitudinal development: Energy of other charged particles.
60	**1592**	Longitudinal development: Energy of other neutral particles.
61	**1705**	Longitudinal development: Energy of e+ and e−
62	**1707**	Longitudinal development: Energy of mu+ and mu−
63	**1711**	Longitudinal development: Energy of pi+ and pi−
64	**1713**	Longitudinal development: Energy of K+ and K−

	Code	*Table name*
65	**1791**	Longitudinal development: Energy of all charged particles.
66	**1792**	Longitudinal development: Energy of all neutral particles.
67	**1793**	Longitudinal development: Energy of all particles.
68	**2001**	Lateral distribution: Gamma rays.
69	**2005**	Lateral distribution: Electrons.
70	**2006**	Lateral distribution: Positrons.
71	**2007**	Lateral distribution: Muons (+).
72	**2008**	Lateral distribution: Muons (−).
73	**2011**	Lateral distribution: Pions (+).
74	**2012**	Lateral distribution: Pions (−).
75	**2013**	Lateral distribution: Kaons (+).
76	**2014**	Lateral distribution: Kaons (−).
77	**2021**	Lateral distribution: Neutrons.
78	**2022**	Lateral distribution: Protons.
79	**2023**	Lateral distribution: Antiprotons.
80	**2041**	Lateral distribution: Nuclei.
81	**2091**	Lateral distribution: Other charged pcles.
82	**2092**	Lateral distribution: Other neutral pcles.
83	**2205**	Lateral distribution: e+ and e−
84	**2207**	Lateral distribution: mu+ and mu−
85	**2211**	Lateral distribution: pi+ and pi−
86	**2213**	Lateral distribution: K+ and K−
87	**2291**	Lateral distribution: All charged particles.
88	**2292**	Lateral distribution: All neutral particles.
89	**2293**	Lateral distribution: All particles.
90	**2301**	Unweighted lateral distribution: Gamma rays.
91	**2305**	Unweighted lateral distribution: Electrons.
92	**2306**	Unweighted lateral distribution: Positrons.
93	**2307**	Unweighted lateral distribution: Muons (+).
94	**2308**	Unweighted lateral distribution: Muons (−).
95	**2311**	Unweighted lateral distribution: Pions (+).
96	**2312**	Unweighted lateral distribution: Pions (−).
97	**2313**	Unweighted lateral distribution: Kaons (+).
98	**2314**	Unweighted lateral distribution: Kaons (−).
99	**2321**	Unweighted lateral distribution: Neutrons.
100	**2322**	Unweighted lateral distribution: Protons.
101	**2323**	Unweighted lateral distribution: Antiprotons.
102	**2341**	Unweighted lateral distribution: Nuclei.
103	**2391**	Unweighted lateral distribution: Other charged pcles.
104	**2392**	Unweighted lateral distribution: Other neutral pcles.
105	**2405**	Unweighted lateral distribution: e+ and e−

	Code	*Table name*
106	**2407**	Unweighted lateral distribution: mu+ and mu−
107	**2411**	Unweighted lateral distribution: pi+ and pi−
108	**2413**	Unweighted lateral distribution: K+ and K−
109	**2491**	Unweighted lateral distribution: All charged particles.
110	**2492**	Unweighted lateral distribution: All neutral particles.
111	**2493**	Unweighted lateral distribution: All particles.
112	**2501**	Energy distribution at ground: Gamma rays.
113	**2505**	Energy distribution at ground: Electrons.
114	**2506**	Energy distribution at ground: Positrons.
115	**2507**	Energy distribution at ground: Muons (+).
116	**2508**	Energy distribution at ground: Muons (−).
117	**2511**	Energy distribution at ground: Pions (+).
118	**2512**	Energy distribution at ground: Pions (−).
119	**2513**	Energy distribution at ground: Kaons (+).
120	**2514**	Energy distribution at ground: Kaons (−).
121	**2521**	Energy distribution at ground: Neutrons.
122	**2522**	Energy distribution at ground: Protons.
123	**2523**	Energy distribution at ground: Antiprotons.
124	**2541**	Energy distribution at ground: Nuclei.
125	**2591**	Energy distribution at ground: Other charged pcles.
126	**2592**	Energy distribution at ground: Other neutral pcles.
127	**2705**	Energy distribution at ground: e+ and e−
128	**2707**	Energy distribution at ground: mu+ and mu−
129	**2711**	Energy distribution at ground: pi+ and pi−
130	**2713**	Energy distribution at ground: K+ and K−
131	**2791**	Energy distribution at ground: All charged particles.
132	**2792**	Energy distribution at ground: All neutral particles.
133	**2793**	Energy distribution at ground: All particles.
134	**2801**	Unweighted energy distribution: Gamma rays.
135	**2805**	Unweighted energy distribution: Electrons.
136	**2806**	Unweighted energy distribution: Positrons.
137	**2807**	Unweighted energy distribution: Muons (+).
138	**2808**	Unweighted energy distribution: Muons (−).
139	**2811**	Unweighted energy distribution: Pions (+).
140	**2812**	Unweighted energy distribution: Pions (−).
141	**2813**	Unweighted energy distribution: Kaons (+).
142	**2814**	Unweighted energy distribution: Kaons (−).
143	**2821**	Unweighted energy distribution: Neutrons.
144	**2822**	Unweighted energy distribution: Protons.
145	**2823**	Unweighted energy distribution: Antiprotons.
146	**2841**	Unweighted energy distribution: Nuclei.

	Code	*Table name*
147	**2891**	Unweighted energy distribution: Other charged pcles.
148	**2892**	Unweighted energy distribution: Other neutral pcles.
149	**2905**	Unweighted energy distribution: e+ and e−
150	**2907**	Unweighted energy distribution: mu+ and mu−
151	**2911**	Unweighted energy distribution: pi+ and pi−
152	**2913**	Unweighted energy distribution: K+ and K−
153	**2991**	Unweighted energy distribution: All charged particles.
154	**2992**	Unweighted energy distribution: All neutral particles.
155	**2993**	Unweighted energy distribution: All particles.
156	**3001**	Mean arrival time distribution: Gamma rays.
157	**3005**	Mean arrival time distribution: Electrons and positrons.
158	**3007**	Mean arrival time distribution: Muons.
159	**3091**	Mean arrival time distribution: Other charged pcles.
160	**3092**	Mean arrival time distribution: Other neutral pcles.
161	**3291**	Mean arrival time distribution: All charged particles.
162	**3292**	Mean arrival time distribution: All neutral particles.
163	**3293**	Mean arrival time distribution: All particles.
164	**5001**	Number and energy of ground gammas versus shower number.
165	**5005**	Number and energy of ground e− versus shower number.
166	**5006**	Number and energy of ground e+ versus shower number.
167	**5007**	Number and energy of ground mu+ versus shower number.
168	**5008**	Number and energy of ground mu− versus shower number.
169	**5011**	Number and energy of ground pi+ versus shower number.
170	**5012**	Number and energy of ground pi− versus shower number.
171	**5013**	Number and energy of ground K+ versus shower number.
172	**5014**	Number and energy of ground K− versus shower number.
173	**5021**	Number and energy of ground neutrons versus shower number.
174	**5022**	Number and energy of ground protons versus shower number.
175	**5023**	Number and energy of ground pbar versus shower number.
176	**5041**	Number and energy of ground nuclei versus shower number.
177	**5091**	Number and energy of other grd. ch. pcles. versus shower number.
178	**5092**	Number and energy of other grd. nt. pcles. versus shower number.
179	**5205**	Number and energy of ground e+ and e− versus shower number.
180	**5207**	Number and energy of ground mu+ and mu− versus shower number.
181	**5211**	Number and energy of ground pi+ and pi− versus shower number.
182	**5213**	Number and energy of ground K+ and K− versus shower number.
183	**5291**	Number and energy of ground ch. pcles. versus shower number.
184	**5292**	Number and energy of ground nt. pcles. versus shower number.
185	**5293**	Number and energy of all ground particles versus shower number.

	Code	*Table name*
186	**5501**	Xmax and Nmax (charged particles) versus shower number.
187	**5511**	First interact. depth and primary energy versus shower number.
188	**5513**	Zenith and azimuth angles versus shower number.
189	**6001**	Number of created particles: Gamma rays.
190	**6005**	Number of created particles: Electrons.
191	**6006**	Number of created particles: Positrons.
192	**6007**	Number of created particles: Muons (+).
193	**6008**	Number of created particles: Muons (-).
194	**6011**	Number of created particles: Pions (+).
195	**6012**	Number of created particles: Pions (-).
196	**6013**	Number of created particles: Kaons (+).
197	**6014**	Number of created particles: Kaons (-).
198	**6021**	Number of created particles: Neutrons.
199	**6022**	Number of created particles: Protons.
200	**6023**	Number of created particles: Antiprotons.
201	**6041**	Number of created particles: Nuclei.
202	**6091**	Number of created particles: Other charged pcles.
203	**6092**	Number of created particles: Other neutral pcles.
204	**6205**	Number of created particles: e+ and e-
205	**6207**	Number of created particles: mu+ and mu-
206	**6211**	Number of created particles: pi+ and pi-
207	**6213**	Number of created particles: K+ and K-
208	**6291**	Number of created particles: All charged particles.
209	**6292**	Number of created particles: All neutral particles.
210	**6293**	Number of created particles: All particles.
211	**6296**	Number of created particles: All neutrinos.
212	**6301**	Number of created entries: Gamma rays.
213	**6305**	Number of created entries: Electrons.
214	**6306**	Number of created entries: Positrons.
215	**6307**	Number of created entries: Muons (+).
216	**6308**	Number of created entries: Muons (-).
217	**6311**	Number of created entries: Pions (+).
218	**6312**	Number of created entries: Pions (-).
219	**6313**	Number of created entries: Kaons (+).
220	**6314**	Number of created entries: Kaons (-).
221	**6321**	Number of created entries: Neutrons.
222	**6322**	Number of created entries: Protons.
223	**6323**	Number of created entries: Antiprotons.
224	**6341**	Number of created entries: Nuclei.
225	**6391**	Number of created entries: Other charged pcles.
226	**6392**	Number of created entries: Other neutral pcles.

	Code	*Table name*
227	**6405**	Number of created entries: e+ and e-
228	**6407**	Number of created entries: mu+ and mu-
229	**6411**	Number of created entries: pi+ and pi-
230	**6413**	Number of created entries: K+ and K-
231	**6491**	Number of created entries: All charged particles.
232	**6492**	Number of created entries: All neutral particles.
233	**6493**	Number of created entries: All particles.
234	**6496**	Number of created entries: All neutrinos.
235	**6501**	Energy of created particles: Gamma rays.
236	**6505**	Energy of created particles: Electrons.
237	**6506**	Energy of created particles: Positrons.
238	**6507**	Energy of created particles: Muons (+).
239	**6508**	Energy of created particles: Muons (-).
240	**6511**	Energy of created particles: Pions (+).
241	**6512**	Energy of created particles: Pions (-).
242	**6513**	Energy of created particles: Kaons (+).
243	**6514**	Energy of created particles: Kaons (-).
244	**6521**	Energy of created particles: Neutrons.
245	**6522**	Energy of created particles: Protons.
246	**6523**	Energy of created particles: Antiprotons.
247	**6541**	Energy of created particles: Nuclei.
248	**6591**	Energy of created particles: Other charged pcles.
249	**6592**	Energy of created particles: Other neutral pcles.
250	**6705**	Energy of created particles: e+ and e-
251	**6707**	Energy of created particles: mu+ and mu-
252	**6711**	Energy of created particles: pi+ and pi-
253	**6713**	Energy of created particles: K+ and K-
254	**6791**	Energy of created particles: All charged particles.
255	**6792**	Energy of created particles: All neutral particles.
256	**6793**	Energy of created particles: All particles.
257	**6796**	Energy of created particles: All neutrinos.
258	**7001**	Longitudinal development: Low energy gamma rays.
259	**7005**	Longitudinal development: Low energy electrons.
260	**7006**	Longitudinal development: Low energy positrons.
261	**7007**	Longitudinal development: Low energy muons (+).
262	**7008**	Longitudinal development: Low energy muons (-).
263	**7091**	Longitudinal development: Other charged low egy. pcles.
264	**7092**	Longitudinal development: Other neutral low egy. pcles.
265	**7205**	Longitudinal development: Low energy e+ and e-
266	**7207**	Longitudinal development: Low energy mu+ and mu-
267	**7291**	Longitudinal development: All low energy charged pcles.

	Code	*Table name*
268	**7292**	Longitudinal development: All low energy neutral pcles.
269	**7293**	Longitudinal development: All low energy pcles.
270	**7301**	Unweighted longit. devel.: Low energy gamma rays.
271	**7305**	Unweighted longit. devel.: Low energy electrons.
272	**7306**	Unweighted longit. devel.: Low energy positrons.
273	**7307**	Unweighted longit. devel.: Low energy muons (+).
274	**7308**	Unweighted longit. devel.: Low energy muons (-).
275	**7391**	Unweighted longit. devel.: Other charged low egy. pcles.
276	**7392**	Unweighted longit. devel.: Other neutral low egy. pcles.
277	**7405**	Unweighted longit. devel.: Low energy e+ and e-
278	**7407**	Unweighted longit. devel.: Low energy mu+ and mu-
279	**7491**	Unweighted longit. devel.: All low energy charged pcles.
280	**7492**	Unweighted longit. devel.: All low energy neutral pcles.
281	**7493**	Unweighted longit. devel.: All low energy pcles.
282	**7501**	Longitudinal development: Energy of low egy. gamma rays.
283	**7505**	Longitudinal development: Energy of low egy. electrons.
284	**7506**	Longitudinal development: Energy of low egy. positrons.
285	**7507**	Longitudinal development: Energy of low egy. muons(+).
286	**7508**	Longitudinal development: Energy of low egy. muons(-).
287	**7591**	Longitudinal development: Egy. of other charged low egy. pcles.
288	**7592**	Longitudinal development: Egy. of other neutral low egy. pcles.
289	**7705**	Longitudinal development: Energy of low egy. e+ and e-
290	**7707**	Longitudinal development: Energy of low egy. mu+ and mu-
291	**7791**	Longitudinal development: Egy. of all low egy. charged pcles.
292	**7792**	Longitudinal development: Egy. of all low egy. neutral pcles.
293	**7793**	Longitudinal development: Egy. of all low energy pcles.
294	**7801**	Longitudinal development: Energy deposited by gamma rays.
295	**7805**	Longitudinal development: Energy deposited by electrons.
296	**7806**	Longitudinal development: Energy deposited by positrons.
297	**7807**	Longitudinal development: Energy deposited by muons (+).
298	**7808**	Longitudinal development: Energy deposited by muons (-).
299	**7891**	Longitudinal development: Egy. deposited by other charged pcles.
300	**7892**	Longitudinal development: Egy. deposited by other neutral pcles.
301	**7905**	Longitudinal development: Energy deposited by e+ and e-
302	**7907**	Longitudinal development: Energy deposited by mu+ and mu-
303	**7991**	Longitudinal development: Egy. deposited by all charged pcles.
304	**7992**	Longitudinal development: Egy. deposited by all neutral pcles.
305	**7993**	Longitudinal development: Energy deposited by all pcles.

Appendix D

The AIRES object library

The AIRES object library is a collection of modules that are useful in several applications, including (but not limited to) special primary modules (see section 3.5), and output file processing, particularly compressed files generated by the AIRES compressed i/o unit (CIO), and many other analysis procedures.

There is an on-line reference for the AIRES library that can be accessed at the following link:

aires.fisica.unlp.edu.ar/doc/aireslibref

The information currently available on the on-line library reference covers all the user-callable library modules and is permanently updated.

D.1 C/C++ interface

The modules of the AIRES object library are callable from a C/C++ program. In general the calling statement is similar to the FORTRAN one, taking into account that all arguments are passed *by reference*. That means that the actual arguments must be pointers to the corresponding data items.

This requirement is made evident when describing the different routines by placing an ampersand (**&**) before the corresponding arguments. The experienced C programmer will understand, however, that this character is not required in actual calling statements containing pointer variables as arguments. The following example illustrates this point:

```
int *channel, *vrb, *irc;
int recnumber;
int crogotorec();
. . .
if (crogotorec(channel, &recnumber, vrb, irc)) { . . .
```

All the arguments of **crogotorec** are defined as pointers, except **recnumber** which is declared as an integer variable. The **&** placed before this argument ensures that this variable be passed by reference to the called routine.

In general, all the FORTRAN routines of the library can be directly called from a C program. In a few cases it was necessary to write special C routines, which were named appending a "c" to the original FORTRAN name, as in the case of **opencrofile** that must be called **opencrofilec** from a C program (see page 196).

It is also worthwhile mentioning that some FORTRAN compilers do place an underscore (_) after the names of the routines. In such cases this character must be manually appended to all the routines used within the C program, excluding, of course, all the special C routines of the previous paragraph.

D.2 List of most frequently used library modules.

In this appendix we list the definitions of the most frequently used routines, alphabetically ordered. At each case the FORTRAN as well as the C calling statements are placed.

atmodelinit

FORTRAN
```
call atmodelinit(modidstr, modparstr, vrb,
                 modname, rc)
```
C
```
atmodelinitc(&modidstr, &modparstr, &vrb,
             &modname, &rc);
```

Initialization of the atmospheric model and specifying its parameters if necessary. This routine is automatically called every time a compressed output file is opened to ensure by default that the atmospheric model used in the analysis is the same as the one in effect during the simulations that originated the compressed data file. For this reason it is very unlikely that a standard analysis application need manual invocation of **atmodelinit**.

Arguments:

modidstr (***Input, character string***) Label to switch among atmospheric models. Maximum length is 16 characters. Current available options are:

Linsley. Linsley's standard model.

SouthPoleAvg. South pole average atmosphere.

LSouthPole. Linsley model for the South pole.

MalargueAvg. Malargue site annual average atmosphere.

GAMMA. J. C. Moreno's model.

Isothermic. Isothermic atmosphere

Homogeneous. Constant density atmosphere.

modparstr (***Input, character string***) String containing model parameters.

vrb (***Input, integer***) Verbosity control. If vrb is zero or negative then no error/informative messages are printed; error conditions are communicated to the calling program via the return code. If vrb is positive error messages will be printed: vrb = 1 means that messages will be printed even with successful operations. vrb = 2,3 means that only error messages will be printed. vrb > 3 is similar to vrb = 3, but with the additional action of stopping the program if a fatal error takes place.

modname (***Output, character string***) A name for the atmospheric model, to be typed somewhere. Maximum length: 42 characters.

rc (***Output, integer***) Return code. Zero means successful return.

atmosinit

FORTRAN `call atmosinit(modlabel, atmosname)`
C `atmosinitc(&modlabel, &atmosname);`

Initialization of the atmospheric parameters. Obsolete routine maintained just to guarantee backwards compatibility of the AIRES system. Presently, this routine initializes the standard Linsley atmosphere, regardless of the model label used.

Arguments:

modlabel (***Input, integer***) Label to switch among atmospheric models. No longer used.

atmosname (***Output, character string***) A name for the atmospheric model. Maximum length: 42 characters.

adstydepth,

adstymdepth FORTRAN `dsty = adstydepth(Xvert, atlayer)`

C `dsty = adstydepth(&Xvert, &atlayer);`
FORTRAN `dsty = adstymdepth(Xvert, atlayer)`
C `dsty = adstymdepth(&Xvert, &atlayer);`

Local density at a given depth. Multilayer atmospheric model. The atmospheric depth is measured in $\mathrm{g/cm^2}$, and the density in $\mathrm{g/cm^3}$ (**adstydepth**) or $\mathrm{m^{-1}g/cm^2}$(**adstydepth**).

Arguments:

Xvert (***Input, double precision***) Vertical atmospheric depth (in $\mathrm{g/cm^2}$) of the corresponding point. Must be positive.

atlayer (***Output, integer***) Atmospheric layer corresponding to the depth **Xvert**. This parameter depends on the selected atmospheric model.

Returned value: (***Double precision***) The local density in $\mathrm{g/cm^3}$ (**adstydepth**) or $\mathrm{m^{-1}g/cm^2}$(**adstydepth**).

cioclose

FORTRAN `call cioclose`
C `cioclose;`

Closing *all* the currently already opened CIO files.

cioclose1

FORTRAN `call cioclose1(channel)`
C `cioclose1(&channel);`

Closing an already opened CIO file.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

ciorinit

FORTRAN `call ciorinit(inilevel, codsys, vrb, irc)`
C `ciorinit(&inilevel, &codsys, &vrb, &irc);`

Initializing the AIRES compressed I/O system for reading data. This routine *must* be invoked at the beginning of every program using the compressed I/O system routines.

Arguments:

inilevel (***Input, integer***) Initialization switch. If inilevel is zero or negative, all needed initialization routines are called. If positive only the CIO system is initialized (The other routines must be called within the invoking program, before calling ciorinit: $\mathbf{inilevel} = 1$ means complete cio initialization, while $\mathbf{inilevel} > 1$ implies only particle coding initialization. This last case allows changing the particle coding system at any moment during a CIO processing session.

codsys (***Input, integer***) Particle coding system identification. This variable permits selecting among several particle coding systems supported by AIRES (see table 4.7). The menu of available systems is the following:

- 0 AIRES internal coding system.
- 1 AIRES internal coding for elementary particles and decimal nuclear notation ($code = A + 100 * Z$).
- 4 Particle Data Group coding system [38] extended with decimal nuclear notation.[1]
- 5 CORSIKA program particle coding system [36].
- 6 GEANT particle coding system [39].
- 8 SIBYLL particle coding system [10], extended with decimal nuclear notation.
- 9 MOCCA style particle coding system, extended with decimal nuclear notation.
- – Any other value is equivalent to $\mathbf{codsys} = 1$.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return. 1 means that an invalid particle coding system was specified by codsys (in this case the default coding system is used).

[1] For nuclei the notation: $code = A + 100000 * Z$, is used.

ciorshutdown

FORTRAN `call ciorshutdown`
C `ciorshutdown;`

Terminating (in an ordered fashion) a compressed file analysis session. This routine should be invoked at the end of every CIO processing program.

clockrandom

FORTRAN `r = clockrandom()`
C `r = clockrandom();`

This function invokes the AIRES elementary random number generator and returns a pseudo-random number uniformly distributed in the interval $(0, 1)$, generated with the current clock and CPU usage lectures. No initialization is needed before using this random number generator.

WARNING: This function is not to be used as a high quality random number generator. This routine is intended only for some special applications like generating a single random seed, for example.

Multiple calls may eventually return correlated numbers if there is no enough time between invocations. Nevertheless, a sequence of different numbers passes direct 1d and 2d chi-square tests, ensuring a minimum quality for the generated numbers.

Returned value: (***Double precision***) The uniform pseudo-random number.

crofieldindex

```
FORTRAN   idx = crofieldindex(channel, rectype, fieldname,
                              vrb, datype, irc)
C         idx = crofieldindex(&channel, &rectype,
                              &fieldname, &vrb, &datype,
                              &irc);
```

Returning the index corresponding to a given field within a compressed file record. It is convenient to use this routine to set integer variables, and use them to manage the data returned by **getcrorecord**, as explained in section 4.2.2 (page 86).

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

rectype (***Input, integer***) Record type (0 for default record type).

fieldname (***Input, character string***) First characters of field name (enough characters must be provided to make an unambiguous specification).

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

datype (***Output, integer***) The data type that corresponds to the specified field: 1 for integer data, 2 for date-time data, and 3 for real data.

irc (***Output, integer***) Return code. 0 means successful return.

Returned value: (***Integer***) The field index. Zero if there was an error.

crofileinfo

FORTRAN `call crofileinfo(channel, ouflag, vrb, irc)`
C `crofileinfo(&channel, &ouflag, &vrb, &irc);`

Printing information about the records of an already opened compressed file. This routine retrieves information about the complete record structure of the corresponding file: How many record types are defined, and for each record type the number of fields and a list of their names and relative logical positions. The ordering in the list of fields is equal to the ordering of data in the integer and real arrays returned by routine **getcrorecord** when reading a record of the same type.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

ouflag (***Integer, input***) Logical output unit(s) selection flag. See routine **croheaderinfo**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

crofileversion

FORTRAN `ivers = crofileversion(channel)`
C `ivers = crofileversion(&channel);`

Returning the AIRES version used to write an already opened compressed file.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

Returned value: (***Integer***) The corresponding version in integer format (for example the number 01040200 for version 1.4.2, 01040201 for version 1.4.2a, etc.). If the file is not opened or if there is an error, then the return value is negative.

crogotorec

FORTRAN
```
okflag = crogotorec(channel, recnumber, vrb,
                    irc)
```
C
```
okflag = crogotorec(&channel, &recnumber, &vrb,
                    &irc);
```

Positioning the file after a given record. This routine, used in connection with **crorecnumber**, allows emulating direct access to compressed files. Notice however that a completely random access regime with very large files may eventually imply longer processing times.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

recnumber (***Input, integer***) The record number. A negative value is taken as zero. If $\mathbf{recnumber} \leq 0$, the return code is always set to zero for successful operations (Notice that in this case the file will be positioned at the beginning of the data records).

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. The meanings of the different values that can be returned are as explained for routine **getcrorecord**. When the return code is a record type, it corresponds to the record type of the *last* scanned record.

Returned value: (***Logical***) True if the positioning was successfully done. False otherwise.

croheaderinfo

FORTRAN `call croheaderinfo(ouflag, vrb, irc)`
C `croheaderinfo(&ouflag, &vrb, &irc);`

Printing a summary of the information contained in the header of the *most recently opened* compressed file.

Arguments:

ouflag (***Input, integer***) Logical output unit(s) selection flag: 0 or negative means FORTRAN unit 6 only, 1 means unit 7 only, 2 means both units 6 and 7, 3 means unit 8 only, $\mathbf{ouflag} > 8$ means unit **ouflag** only. FORTRAN unit 6 corresponds to the standard output channel.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

croinputdata0

FORTRAN
```
call croinputdata0(intdata, realdata, shprimcode,
                   shprimwt)
```
C
```
croinputdata0(&intdata[1], &realdata[1],
              &shprimcode[1], &shprimwt[1]);
```

Copying into arrays some header data items corresponding to the *most recently opened* compressed file. Notice that some additional input parameters must be retrieved using routines **getinpreal**, **getinpint** or **getinpswitch** (see pages 176–179).

Arguments:

intdata (***Output, integer, array(*)***) Integer data array. The calling program must provide enough space for it. The following list describes the different data items:

1	Number of different primary particles.
2-4	Reserved for future use.
5	Primary energy distribution: 0 fixed energy; 1 varying energy.
6	Zenith angle distribution: 0, fixed angle; 1, *sine* distribution (equation (3.4)) ; 2, *sine-cosine* distribution (equation (3.6)).
7	Azimuth angle distribution: 0 (10), fixed angle (geographic azimuth); 1 (11), varying angle (geographic azimuths).
8	Number of observing levels.
9	Atmospheric model label (See page 109).
10-14	Reserved for future use.
15	First shower number.

realdata (***Output, double precision, array(*)***) Real data array. The calling program must provide enough space for it. The following list describes the different data items:

1	Minimum primary energy (GeV).
2	Maximum primary energy (GeV).
3	Exponent γ of energy distribution (equation (3.2)).
4	Minimum zenith angle (deg).
5	Maximum zenith angle (deg).
6	Minimum azimuth angle (deg).
7	Maximum azimuth angle (deg).
8	Thinning energy parameter.[2]

[2]The thinning energy parameter, t_p, must be interpreted as follows: When positive, it gives the absolute thinning energy in GeV. Otherwise it indicates a relative thinning specification, being $E_{\rm th} = |t_p| E_{\rm primary}$.

9 Injection altitude (m).[3]
10 Injection depth (g/cm^2).
11 Ground altitude (m).
12 Ground depth (g/cm^2).
13-14 Reserved for future use.
15 Altitude of first observing level (m).
16 Vertical depth of first observing level (g/cm^2).
17 Altitude of last observing level (m).
18 Vertical depth of last observing level (g/cm^2).
19 Distance between consecutive observing levels in g/cm^2.
20 Site latitude (deg).
21 Site longitude (deg).
22 Geomagnetic field strength, F, (nT).
23 Local geomagnetic inclination, I, (deg).
24 Local geomagnetic declination, D, (deg).
25 Amplitude of random fluctuation of magnetic field[4].
26-29 Reserved for future use.
30 Minimum lateral distance used for ground particle histograms (m).
31 Maximum lateral distance used for ground particle histograms (m).
32 Minimum energy used for histograms (GeV).
33 Maximum energy used for histograms (GeV).
34-35 Reserved for future use.
36 Minimum radial distance parameter for the *most recently opened* compressed file (m).
37 Maximum radial distance parameter for the *most recently opened* compressed file (m).

shprimcode (***Output, integer, array(*)***) For i from 1 to $\mathbf{intdata}(1)$, $\mathbf{shprimcode}(i)$ gives the corresponding primary particle code. The coding system used is the one defined when starting the cio system.

shprimwt (***Output, double precision, array(*)***) For i from 1 to $\mathbf{intdata}(1)$, $\mathbf{shprimwt}(i)$ gives the corresponding primary particle weight. This weight is 1 in the single primary case.

[3]Measured vertically, starting from the intersection point between the sea level and the line that goes form the Earth's center to the particle injection point, i.e., z_v in figure 2.1.

[4]The magnetic fluctuation parameter, f_p, must be interpreted as follows: When positive, it gives the absolute fluctuation in nT. Otherwise it indicates a relative fluctuation specification, being $\Delta B = |f_p|\,\mathrm{F}$.

crooldata

FORTRAN
```
call crooldata(vrb, nobslev, olzv, oldepth,
                    irc)
```
C
```
crooldata(&vrb, &nobslev, &olzv[1],
          &oldepth[1], &irc);
```

Calculating observing levels information from data contained in a compressed data file header.

Since the header data is of global nature, the data used by this routine corresponds to the *most recently opened* compressed file.

Arguments:

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

nobslev (***Output, integer***) The number of observing levels.

olzv (***Output, double precision, array(*)***) Altitudes (in m) of the corresponding observing levels, from 1 to **nobslev**. The calling program must ensure that there is enough space for this array.

oldepth (***Output, double precision, array(*)***) Vertical atmospheric depth (in $\mathrm{g/cm^2}$) of the corresponding observing levels, from 1 to **nobslev**. The calling program must ensure that there is enough space for this array.

irc (***Output, integer***) Return code. 0 means successful return.

croreccount

FORTRAN
```
call croreccount(channel, vrb, nrtype, nrec, irc)
```
C
```
croreccount(&channel, &vrb, &nrtype[0], &nrec,
            &irc);
```

Counting the records of a compressed file starting from the first non-read record. Once the file was scanned, the corresponding I/O channel is left in "end of file" status.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

nrtype (***Output, integer***) The highest record type defined for the file (record types range from zero to **nrtype**).

nrec (***Output, integer, array(0:`nrtype`)***) For each record type, the number of records found. No check is made to ensure that the length of the array is enough to store all the data items.

irc (***Output, integer***) Return code. 0 means successful return.

crorecfind

FORTRAN
```
okflag = crorecfind(channel, intype, vrb,
                    infield1, rectype)
```
C
```
okflag = crorecfind(&channel, &intype, &vrb,
                    &infield1, &rectype);
```

Reading records until getting a specified record type. The compressed file associated with **channel** is scanned until a record of type **intype** is found.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

intype (***Input, integer***) Record type to find.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

infield1 (***Output, integer***) If **intype** is zero, this variable contains the current value of the first integer field of the last scanned record (which will be, in general, a particle code). Otherwise it is set to zero.

rectype (***Output, integer***) Last scanned record type and return code. This argument contains the same information as argument **irc** of routine **getcrorecord**. Notice that in the case of successful return, **rectype** is equal to **intype**.

Returned value: (***Logical***) True if the last record was successfully read. False otherwise (End of file or I/O error).

crorecinfo

FORTRAN
```
call crorecninfo(channel, poskey, ouflag, vrb,
                 irc)
```
C
```
crorecninfo(&channel, &poskey, &ouflag, &vrb,
            &irc);
```

Printing information about the total number of records within an already opened compressed file. The file is scanned starting after the last record already read to count the number of records of each type that were written into it.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

poskey (***Input, integer***) Positioning key. This parameter allows to control the file positioning after returning from this routine: If zero or negative the file remains positioned at the "end of file" point, if 1 at the beginning of data, and if greater than 1, at the position found before the call (This last option may eventually imply a significant increase in processing time for very large files).

ouflag (***Integer, input***) Logical output unit(s) selection flag. See routine **croheaderinfo**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

crorecnumber

FORTRAN
```
recno = crorecnumber(channel, vrb, irc)
```
C
```
recno = crorecnumber(&channel, &vrb, &irc);
```

This function returns the current record number corresponding to an already opened compressed file.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

Returned value: (***Integer***) The record number. If the file is not ready (closed or end of file), then -1 is returned.

crorecstrut

FORTRAN
```
call crorecstruct(channel, nrtype, nintf, nrealf,
                  irc)
```
C
```
crorecstruct(&channel, &nrtype, &nintf, &nrealf,
             &irc);
```

Getting information about the records of an already opened compressed file.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

nrtype (***Output, integer***) The highest record type defined for the file (record types range from zero to **nrtype**).

nintf (***Output, integer, array(0:`nrtype`)***) Number of integer fields contained at each record type, for record types from zero to **nrtype**. No check is made to ensure that the length of the array is enough to store all the data items.

nrealf (***Output, integer, array(0:`nrtype`)***) Number of real fields contained at each record type, for record types from zero to **nrtype**. No check is made to ensure that the length of the array is enough to store all the data items.

irc (***Output, integer***) Return code. 0 means successful return.

crorewind

FORTRAN `call crorewind(channel, vrb, irc)`
C `crorewind(&channel, &vrb, &irc);`

"Rewinding" an already opened compressed file. The file is positioned just before the first data record. In other words, the file is system rewound and its header is re-scanned so the file pointer remains located at the beginning of the record data stream.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

crospcode

FORTRAN `isspecial = crospcode(pcode, splabel)`
C `isspecial = crospcode(&pcode, &splabel);`

This logical function determines whether or not a given particle code corresponds to a special primary particle.

Arguments:

pcode (***Input, integer***) The particle code to check.

splabel (***Output, integer***) Label associated to the special particle, or zero if the code does not corresponds to a special particle. This variable is useful for further use with other library routines, and should not be set by the calling program.

Returned value: (***Logical***) True if the input code corresponds to a special primary particle. False otherwise.

crospmodinfo

FORTRAN
```
call crospmodinfo(spname, spmodu, spml, sppars,
                  sppl, irc)
```
C
```
crospmodinfoc(&spname, &spmodu, &spml, &sppars,
              &sppl, &irc);
```

Retrieving information about the external module associated to a already defined special particle. When this routine is used to retrieve information stored in a compressed file, the data returned correspond to the *most recently opened* compressed file.

Arguments:

intdata (***Input, string***) The name of the special particle.

spmodu (***Output, string***) The name of the associated module. The calling program must provide enough space for this string.

spmodu (***Output, integer***) Length of string **spmodu**.

sppars (***Output, string***) String containing the parameters passed to the module. The calling program must provide enough space for this string.

sppl (***Output, integer***) Length of string **sppars**.

crospnames

FORTRAN `call crospnames(nspp, spname)`
C `crospnamesc(&nspp, &spname[1]);`

Retrieving the names of the currently defined special particles. When this routine is used to retrieve information stored in a compressed file, the data returned correspond to the *most recently opened* compressed file.

Arguments:

nspp (***Input, integer***) The number of special particles defined.

spname (***Output, string, array(*)***) Array containing the names of the defined particles. The calling program must provide enough space for this array, and its elements (maximum 16 characters each).

crotaskid

FORTRAN
```
call crotaskid(taskname, tasknamelen,
                    taskversion, startdate)
```
C
```
crotaskidc(&taskname, &tasknamelen,
                &taskversion, &startdate);
```

Getting task name and starting date for the task corresponding to the *most recently opened* compressed file.

Arguments:

taskname (***Output, string***) The task name. The calling program must ensure there is enough space to store the string.

tasknamelen (***Output, integer***) Length of task name.

tasknameversion (***Output, integer***) Task version.

startdate (***Output, string***) Task starting date in the format “dd/Mmm/yyyy hh:mm:ss” (20 characters).

dati

FORTRAN `call dati(datistr)`
C `dati(&datistr);`

Current date and time in the format dd/Mmm/yyyy hh:mm:ss

Arguments:

datistr (***Output, character string***) The string containing the current date and time.

depthfromz

FORTRAN `Xvert = depthfromz(z, atlayer)`
C `Xvert = depthfromz(&z, &atlayer);`

Atmospheric depth from vertical altitude. Multilayer atmospheric model..

Arguments:

z (***Input, double precision***) The vertical altitude in meters above sea level.

atlayer (***Output, integer***) Atmospheric layer corresponding to the depth **Xvert**. This parameter depends on the selected atmospheric model.

Returned value: (***Double precision***) The vertical depth in g/cm^2

dumpfileversion

FORTRAN `ivers = dumpfileversion()`
C `ivers = dumpfileversion();`

Returning the AIRES version associated with the dump file that was *most recently read in* (this can be done using routine **loadumpfile**).

Returned value: (***Integer***) The corresponding version in integer format (for example the number 01040200 for version 1.4.2, 01040201 for version 1.4.2a, etc.). If there is an error, then the return value is negative.

dumpfileversiono

FORTRAN `ivers = dumpfileversiono()`
C `ivers = dumpfileversiono();`

Returning the AIRES version used to write for the first time (the original version) the dump file that was *most recently read in* (this can be done using routine **loadumpfile**).

Returned value: (***Integer***) The corresponding version in integer format (for example the number 01040200 for version 1.4.2, 01040201 for version 1.4.2a, etc.). If there is an error, then the return value is negative.

dumpinputdata0

FORTRAN `call dumpinputdata0(intdata, realdata)`
C `dumpinputdata0(&intdata[1], &realdata[1]);`

Copying into arrays some global input data parameters stored in the dump file that was *most recently read in* (this can be done using routine **loadumpfile**), that are not returned by **croinputdata0**.

Arguments:

intdata (***Output, integer, array(*)***) Integer data array. The calling program must provide enough space for it. The following list describes the different data items:

1	Total number of showers.
2	Number of completed showers.
3	First shower number.
4-9	Reserved for future use.
10	Separate showers integer parameter.

realdata (***Output, double precision, array(*)***) Real data array. The calling program must provide enough space for it. The following list describes the different data items:

1-	Reserved for future use.

fitghf

FORTRAN
```
call fitghf(bodata0, eodata0, depths,
                    nallch, weights, ws, minnmax,
                    nminratio, bodataeff, eodataeff,
                    nmax, xmax, x0, lambda, sqsum,
                    irc)
```
C
```
fitghf(&bodata0, &eodata0, &depths[1],
       &nallch[1], &weights[1], &ws,
       &minnmax, &nminratio, &bodataeff,
       &eodataeff, &nmax, &xmax, &x0,
       &lambda, &sqsum, &irc);
```

Performing a 4-parameter nonlinear least squares fit to evaluate the parameters $N_{\max}$, $X_{\max}$, X_0 and λ of the Gaisser-Hillas function of equation (4.1). The fit is done using the Levenberg-Mardquardt algorithm, as implemented in the public domain software library Netlib [16].

Arguments:

bodata0, eodata0 (***Input, integer***) Positive integer parameters defining the number of data points to use in the fit.

depths (***Input, double precision, array(eodata0)***) Depths of the observing levels used in the fit. Only the range (**bodata0**:**eodata0**) is used.

nallch (***Input, double precision, array(eodata0)***) Number of charged particles crossing the different levels. Only the range (**bodata0**:**eodata0**) is used.

weights (***Input, double precision, array(eodata0)***) Positive weights to be assigned to each one of the data points. Only the range (**bodata0**:**eodata0**) is used.

ws (***Input, integer***) If $\mathbf{ws} = 2$, the weights are evaluated internally (proportionally to the square root of the number of particles). If $\mathbf{ws} = 1$ they must be provided as input data. If $\mathbf{ws} = 2$ the array **weights** is not used.

minnmax (***Input, double precision***) Threshold value for the maximum number of particles in the input data set. The fit is not performed if the maximum number of particles is below this parameter. If **minnmax** is negative, it is taken as zero.

nminratio (***Input, double precision***) Positive parameter used to determine the end of the data set. Must be equal or greater than 5. Once the maximum of the data set is found. the points located after this maximum up to the point where the number of charged particles is less than the maximum divided **nminratio**. The remaining part of the data is not taken into account in the fit. A similar analysis is performed with the points located before the maximum. The recommended value is 100. A very large value will enforce inclusion of all the data set.

bodataeff, eodataeff (***Output, integer***) The actual range of data points used in the fit.

nmax (***Output, double precision***) Estimated number of charged particles at the shower maximum (parameter $N_{\max}$). If no fit was possible, then the value coming from a direct estimation from the input data is returned.

xmax (***Output, double precision***) Fitted position of the shower maximum, $X_{\max}$, in $\mathrm{g/cm^2}$. If no fit was possible, then the value coming from a direct estimation from the input data is returned.

x0 (***Output, double precision***) Fitted position of the point where the Gaisser-Hillas function is zero (parameter X_0), expressed in $\mathrm{g/cm^2}$.

lambda (***Output, double precision***) Fitted parameter λ, in $\mathrm{g/cm^2}$.

sqsum (***Output, double precision***) The resulting normalized sum of squares:

$$S = \frac{1}{N\,N_{\max}} \sum_{i=1}^{N} \frac{\left[N^{(I)}(i) - N^{(GH)}(i)\right]^2}{N^{(GH)}(i)}, \tag{D.1}$$

where N is the number of data points used in the fit, and $N^{(I)}$ ($N^{(GH)}$) represent the set of particle numbers given as input (returned from equation 4.1 for the corresponding depths).

irc (***Output, integer***) Return code. Zero means that the fit was successfully completed.

getcrorecord

FORTRAN
```
okflag = getcrorecord(channel, intfields,
                      realfields, altrec, vrb,
                      irc)
```
C
```
okflag = getcrorecord(&channel, &intfields[1],
                      &realfields[1], &altrec,
                      &vrb, &irc);
```

Reading a record from a compressed data file already opened. This routine can be used to read records from *every* kind of compressed file: The routine automatically processes the records without needing any user-level specification beyond file identity (parameter **channel**). The logical returned value (here assigned to logical variable **okflag**) permits determining whether or not the read operation was successful. The characteristics of the read record are informed via the return code (**irc**), and the arrays **intfields** and **realfields** contain the corresponding data items. Their contents depend on the file being processed and on the record type. The auxiliary routines **crofileinfo** and **crofieldindex** are useful to process adequately the returned data at each case.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

intfields (***Output, integer, array(*)***) Integer fields of the last read record. This includes the non-scaled integer quantities and (in the last positions) the date-time specification(s), if any. The calling program must provide enough space for this array (The minimum dimension is the maximum number of fields that can appear in a record plus 1). Positions beyond the last integer fields are used as scratch working space. The meaning of each data item within this array varies with the class of file processed and with the record type (see also argument **irc** and routine **crofileinfo**).

realfields (***Output, double precision, array(*)***) Real fields of the record. The calling program must provide enough space for this array. The meaning of each data item within this array varies with the class of file processed and with the record type (see also argument **irc** and routine **crofileinfo**).

altrec (***Output, logical***) True if the corresponding record type is positive (alternative record type) False if the record type is zero (default record type).

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only

error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means that a record with zero (default) record type was successfully read. i ($i > 0$) means that an alternative record of type i was successfully read. -1 means that an end-of-file condition was got from the corresponding file. Any other value indicates a reading error (**irc** equals the system return code plus 10000).

Returned value: (***Logical***) True if a record was successfully read. False otherwise (End of file or I/O error).

getcrorectype

```
FORTRAN   okflag = getcrorectype(channel, vrb, infield1,
                                 rectype)
C         okflag = getcrorectype(&channel, &vrb, &infield1,
                                 &rectype);
```

Getting the record type of the record which is located next to the last read record of the compressed file identified by argument **channel**.

The action of this routine consists in reading the first part of the record to obtain the record type, and then skip the remaining part to position the file at the end of the corresponding record. The use of this routine is recommended whenever only the record type is needed, since it is faster than **getcrorecord**. When additional data of an already scanned record is required, routine **regetcrorecord** can be used to re-scan the last processed one.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

infield1 (***Output, integer***) If **rectype** is zero, this variable contains the current value of the first integer field of the record (which is, in general, a particle code). Otherwise it is set to zero.

rectype (***Output, integer***) Record type and return code. This argument contains the same information as argument **irc** of routine **getcrorecord**.

Returned value: (***Logical***) True if a record was successfully read. False otherwise (End of file or I/O error).

getglobal

FORTRAN `call getglobal(gvname, sdynsw, gvval, valen)`
C `getglobalc(&gvname, &sdynsw, &gvval, &valen);`

Getting the current value of an already defined global varible. When this routine is used to retrieve information stored in a compressed file, the data returned correspond to the *most recently opened* compressed file.

Arguments:

gvname (***Input, string***) Name of global variable.

sdynsw (***Output, integer***) Type of variable: 1 dynamic, 2 static, 0 if the variable is undefined.

gvval (***Output, string***) The string currently assigned to the variable. The calling program must ensure enough space to store the string.

valen (***Output, integer***) Length of **gvval**. **valen** is negative for undefined variables.

getinpint

FORTRAN `value = getinpint(dirname)`
C `value = getinpintc(&dirname);`

Getting the current value for an **integer** (static) input parameter corresponding to the *most recently opened* compressed file. This routine is used to get from the current file's header those integer input parameters not returned by routine **croinputdata0** (see page 152).

Arguments:

dirname (***Input, string***) Name of the IDL directive associated with the parameter (can be abbreviated accordingly with the rules described in appendix B).

Returned value: (***integer***) The current setting for the corresponding parameter. In case of error the returned value is undefined.

getinpreal

FORTRAN `value = getinpreal(dirname)`
C `value = getinprealc(&dirname);`

Getting the current value for a **real** (static) input parameter corresponding to the *most recently opened* compressed file. This routine is used to get from the current file's header those real input parameters not returned by routine **croinputdata0** (see page 152).

Arguments:

dirname (***Input, string***) Name of the IDL directive associated with the parameter (can be abbreviated accordingly with the rules described in appendix B).

Returned value: (***double precision***) The current setting for the corresponding parameter. In case of error the returned value is undefined.

getinpstring

FORTRAN `call getinpstring(dirname, value, slen)`
C `getinpstringc(&dirname, &value, &slen);`

Getting the current value for an input (static) **character string** corresponding to the *most recently opened* compressed file.

Arguments:

dirname (***Input, string***) Name of the IDL directive associated with the parameter (can be abbreviated accordingly with the rules described in appendix B).

value (***Output, string***) The current parameter value. The calling program must ensure that there is enough space to store the string.

slen (***Output, integer***) Length of the current parameter value. On error, **slen** is negative.

getinpswitch

FORTRAN `value = getinpswitch(dirname)`
C `value = getinpswitchc(&dirname);`

Getting the current value for an input (static) **logical switch** corresponding to the *most recently opened* compressed file. This routine is used to get from the current file's header those logical input parameters not returned by routine **croinputdata0** (see page 152).

Arguments:

dirname (***Input, string***) Name of the IDL directive associated with the parameter (can be abbreviated accordingly with the rules described in appendix B).

Returned value: (***Logical***) The current setting for the corresponding parameter. In case of error the returned value is undefined.

getlgtinit

FORTRAN `call getlgtinit(channel, vrb, irc)`
C `getlgtinit(&channel, &vrb, &irc);`

Initializing internal data needed to process records from compressed longitudinal particle tracking files by means of routine **getlgtrecord** and related ones. This routine should be called immediately after opening the corresponding compressed file.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

getlgtrecord

FORTRAN
```
okflag = getlgtrecord(channel, currol, updown,
                      intfields, realfields,
                      altrec, vrb, irc)
```
C
```
okflag = getlgtrecord(&channel, &currol, &updown,
                      &intfields[1],
                      &realfields[1], &altrec,
                      &vrb, &irc);
```

Reading a record from a compressed longitudinal particle tracking file and returning the read data in a "level per level" basis. This routine invokes **getcrorecord** to get a record from the corresponding compressed file when it is necessary, and must be used jointly with **getlgtinit**.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

currol (***Output, integer***) Observing level crossed by the particle.

updown (***Output, integer***) Up-down indicator: 1 if the particle is going upwards, -1 otherwise.

intfields (***Output, integer, array(*)***) Integer fields of the last read record. This includes the non-scaled integer quantities and (in the last positions) the date-time specification(s), if any. The calling program must provide enough space for this array (The minimum dimension is the maximum number of fields that can appear in a record plus 1). Positions beyond the last integer fields are used as scratch working space. The meaning of each data item within this array varies with the class of file processed and with the record type (see also argument **irc** and routine **crofileinfo**).

realfields (***Output, double precision, array(*)***) Real fields of the record. The calling program must provide enough space for this array. The meaning of each data item within this array varies with the class of file processed and with the record type (see also argument **irc** and routine **crofileinfo**).

altrec (***Output, logical***) True if the corresponding record type is positive (alternative record type) False if the record type is zero (default record type).

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only

error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means that a record with zero (default) record type was successfully read. i ($i > 0$) means that an alternative record of type i was successfully read. -1 means that an end-of-file condition was got from the corresponding file. Any other value indicates a reading error (**irc** equals the system return code plus 10000).

Returned value: (***Logical***) True if a record was successfully read. False otherwise (End of file or I/O error).

ghfpars

FORTRAN
```
call ghfpars(nmax, xmax, x0, lambda, vrb,
                    irc)
```
C
```
ghfpars(&nmax, &xmax, &x0, &lambda, &vrb,
        &irc);
```

Setting the internal quantities needed to work with the Gaisser-Hillas function (equation (4.1)) related routines.

Arguments:

nmax (***Input, double precision***) Parameter $N_{\max}$ of equation 4.1.

xmax (***Input, double precision***) Parameter $X_{\max}$ of equation 4.1.

x0 (***Input, double precision***) Parameter X_0 of equation 4.1.

lambda (***Input, double precision***) Parameter λ of equation 4.1.

irc (***Output, integer***) Return code. 0 means successful return.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

ghfin

FORTRAN `x = ghfin(np, prepost)`
C `x = ghfin(&np, &prepost);`

Numerical evaluation of the inverse of the Gaisser-Hillas function (equation (4.1)) for a given number of particles **np**. The four parameters $N_{\max}$, $X_{\max}$, X_0, and λ must be specified previously by means of **ghfpars**.

Arguments:

np (***Input, double precision***) The number of particles. If $\mathbf{np} < 0$ or $\mathbf{np} > N_{\max}$, the result is a large negative number.

prepost (***Input, integer***) Integer parameter labeling which of the two abscissas X has to be returned: If prepost is less or equal to 0 then $X < X_{\max}$; otherwise $X > X_{\max}$. Notice that the inverse of the Gaisser-Hillas function is bi-valuated.

Returned value: (***Double precision***) The value of the inverse Gaisser-Hillas function, expressed in $\mathrm{g/cm^2}$, that is, **x** such that **np = ghfx(x)**.

ghfx

FORTRAN `np = ghfx(x)`
C `np = ghfx(&x);`

Evaluating the Gaisser-Hillas function (equation (4.1)) for a given depth **x**. The four parameters $N_{\max}$, $X_{\max}$, X_0, and λ must be specified previously by means of **ghfpars**.

Arguments:

x (***Input, double precision***) Atmospheric depth in $\mathrm{g/cm^2}$.

Returned value: (***Double precision***) The value of the function at the specified **x**.

grandom

FORTRAN `r = grandom()`
C `r = grandom();`

This function invokes the AIRES random number generator and returns a pseudo-random number with normal Gaussian distribution (zero mean and unit standard deviation). It is necessary to initialize the random series calling **raninit** before using this function.

Returned value: (***Double precision***) The Gaussian pseudo-random number.

idlcheck

FORTRAN `ikey = idlcheck(dirname)`
C `ikey = idlcheckc(&dirname);`

Checking a string to see if it matches any of the IDL instructions currently defined, that is, the ones corresponding to the *most recently opened* compressed file.

Arguments:

dirname (***Input, string***) Name of the IDL directive to be checked (can be abbreviated accordingly with the rules described in appendix B).

Returned value: (***Integer***) If an error occurs, then the returned value will be negative. Other return values are the following:

- 0 The string does not match any of the currently valid IDL instructions.
- 1 The string matches a directive belonging to the "basic" instruction set with no parameter(s) associated with it, for example **Help**.
- 2 The string matches a directive belonging to the "basic" instruction set. If there is a parameter associated with the directive, then it can be obtained by means of routine **croinputdata0**.
- 4 The directive corresponds to a real input parameter. The parameter can be retrieved by means of function **getinpreal**.
- 6 The directive corresponds to an integer input parameter. The parameter can be retrieved by means of function **getinpint**.
- 8 The directive corresponds to a logical input parameter. The parameter can be retrieved by means of function **getinpswitch**.
- 10 The directive correspond to a string input parameter. The parameter can be retrieved by means of routine **getinpstring**.

loadumpfile

FORTRAN `call loadumpfile(wdir, taskname, vrb, irc)`
C `loadumpfilec(&wdir, &taskname, &vrb, &irc);`

Reading the dump file associated with a given task, and copying into internal variables all the information contained within it.

Arguments:

wdir (***Input, character string***) The name of the directory where the file is placed. It defaults to the current directory when blank.

taskname (***Input, character string***) Task name, or dump file name.

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return. 1 means successful return, but the dump file was not created using the same AIRES version. 8 means that no dump file (in the sequence **taskname**, **taskname.adf**, **taskname.idf**) exists. 12 means invalid file name. Other return codes come from the adf or idf read routines.

nuclcode

FORTRAN `ncode = nuclcode(z, n, irc)`
C `ncode = nuclcode(&z, &n, &irc);`

This routine returns the AIRES code of a nucleus of Z protons and N neutrons, as defined in page 18.

Arguments:

z (***Input, integer***) The number of protons in the nucleus.

n (***Input, integer***) The number of neutrons in the nucleus.

irc (***Output, integer***) Return code. 0 means that a valid pair of input parameters (Z, N) was successfully processed. 3 means that the nucleus cannot be specified with the AIRES system. 5 means that either Z or N are out of allowed ranges.

Returned value: (***Integer***) The nucleus code of equation (2.17).

nucldecode

FORTRAN `call nucldecode(ncode, z, n, a)`
C `nucldecode(&ncode, &z, &n, &a);`

This routine returns the charge, neutron and mass numbers corresponding to a given AIRES nuclear code (see page 18).

Arguments:

ncode (***Input, integer***) The AIRES nuclear code of equation (2.17).

z (***Output, integer***) The number of protons in the nucleus.

n (***Output, integer***) The number of neutrons in the nucleus.

a (***Output, integer***) The mass number.

olcoord

FORTRAN
```
call olcoord(nobslev, olzv, groundz, injz,
             zenith, azimuth, xaxis, yaxis,
             zaxis, tshift, mx, my, irc)
```
C
```
olcoord(&nobslev, &olzv[1], &groundz,
        &injz, &zenith, &azimuth,
        &xaxis[1], &yaxis[1], &zaxis[1],
        &tshift[1], &mx[1], &my[1], &irc);
```

This routine evaluates the coordinates of the intersections of observing level surfaces with the shower axis, (x_{0i}, y_{0i}, z_{0i}), $i = 1, \ldots, N_o$, the corresponding time shifts, t_{0i}, and the coefficients, m_{xi}, m_{yi}, of the plane tangent to the surface at the intersection point:

$$z - z_{0i} = m_{xi}(x - x_{0i}) + m_{yi}(y - y_{0i}), \quad i = 1, \ldots, N_o. \tag{D.2}$$

Arguments:

nobslev (***Input, integer***) The number of observing levels (N_o).

olzv (***Input, double precision, array(*`nobslev`*)***) Altitudes (in m) of the corresponding observing levels.

groundz (***Input, double precision***) Ground altitude (in m).

injz (***Input, double precision***) Injection altitude (in m).

zenith (***Input, double precision***) Shower zenith angle (deg).

azimuth (***Input, double precision***) Shower azimuth angle (deg).

xaxis, yaxis, zaxis (***Output, double precision, array(*`nobslev`*)***) Respectively x_{0i}, y_{0i} and z_{0i}, $i = 1, \ldots, N_o$, coordinates (in m) of the intersection points between the observing level surfaces and the shower axis.

tshift (***Output, double precision, array(*`nobslev`*)***) Observing levels time shifts, t_{0i}, $i = 1, \ldots, N_o$, (in ns), that is, the amount of time a particle moving at the speed of light needs to go from the shower injection point to corresponding intersection point (x_{0i}, y_{0i}, z_{0i}).

mx, my (***Output, double precision, array(*`nobslev`*)***) Coefficients of the planes which are tangent to the observing levels and pass by the corresponding intersection points.

irc (***Output, integer***) Return code. Zero means successful return.

olcrossed

FORTRAN
```
call olcrossed(olkey, updown, firstol, lastol)
```
C
```
olcrossed(&olkey, &updown, &firstol,
          &lastol);
```

This routine reconstructs the information contained in the *crossed observing levels key,* one of the data items saved at each particle record in any longitudinal tracking compressed file.

This key encodes the first and last crossed observing observing levels and the direction of motion. The encoding formula defined in equation (4.6), where L, i_f and i_l correspond to **olkey**, **firstol** and **lastol**, respectively.

The routine returns all the variables of the right hand side of equation (4.6). The variable associated to $s_{\rm ud}$, **updown** is set in a slightly different way: It is be set to 1 when the particle goes upwards, and to -1 otherwise.

Arguments:

olkey (***Input, integer***) Key with information about the crossed observing levels.

updown (***Output, integer***) Up-down indicator: 1 if the particle is going upwards, -1 otherwise.

firstol (***Output, integer***) First observing level crossed ($1 \leq \mathbf{firstol} \leq 510$).

lastol (***Output, integer***) Last observing level crossed ($1 \leq \mathbf{lastol} \leq 510$).

olcrossedu

FORTRAN
```
call olcrossedu(olkey, ux, uy, uz, firstol,
                lastol)
```
C
```
olcrossedu(&olkey, &ux, &uy, &uz, &firstol,
           &lastol);
```

This routine is similar to **olcrossed**, but retrieves the information about the particle's direction of motion (up or down) in the form of an unitary vector.

Arguments:

olkey (***Input, integer***) Key with information about the crossed observing levels (See routine **olcrossed**).

ux, uy (***Input, double precision***) x and y components of the unitary vector marking the particle's direction of motion.

uz (***Output, double precision***) z component of the direction of motion. Positive means upwards motion.

firstol (***Output, integer***) First observing level crossed ($1 \leq \mathbf{firstol} \leq 510$).

lastol (***Output, integer***) Last observing level crossed ($1 \leq \mathbf{lastol} \leq 510$).

olsavemarked

FORTRAN `ismarked = olsavemarked(obslev, vrb, irc)`
C `ismarked = olsavemarked(&obslev, &vrb, &irc);`

Logical function returning "true" if an observing level is marked to be saved into longitudinal files, "false" otherwise. An arbitrary subset of the defined observing levels can be selected for inclusion into the longitudinal compressed files (see page 125); this function allows to determine if a given observing level was or not marked at the moment of performing the simulations that generated the corresponding compressed file.

Arguments:

obslev (***Input, integer***) The number of observing level. If it is out of range the returned value will always be "false".

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. 0 means successful return.

Returned value: (***Logical***) "true" if the level is marked for file recording, "false" otherwise.

olv2slant

FORTRAN
```
call olv2slant(nobslev, olxv, Xv0, zendis,
                        zen1, zen2, groundz, olxs)
```
C
```
olv2slant(&nobslev, &olxv[1], &Xv0, &zendis,
                  &zen1, &zen2, &groundz, &olxs[1]);
```

Evaluating the slant depths of a set of observing levels. The slant depths are calculated along an axis starting at altitude **zground**, for the "segment" that ends at vertical depth **Xv0** (**Xv0** $= 0$ is the top of the atmosphere). The integer variable **zendis** allows to select among fixed, *sine* and *sine-cosine* zenith angle distributions (see section 3.3.3).

Arguments:

nobslev (***Input, integer***) The number of observing levels (N_o).

olxv (***Input, double precision, array(`nobslev`)***) Vertical atmospheric depths (in $\mathrm{g/cm^2}$) of the corresponding observing levels.

Xv0 (***Input, double precision***) Vertical atmospheric depth (in $\mathrm{g/cm^2}$) of the point marking the end of the integration path. If **Xv0** is zero, then the end of the integration path is the top of the atmosphere.

zendis (***Input, integer***) Zenith angle distribution switch: 0 – fixed zenith angle, 1 – sine distribution, 2 – sine-cosine distribution.

zen1, zen2 (***Input, double precision***) Minimum and maximum zenith angles (degrees). If **zendis** is 0, then **zen2** is not used and **zen1** gives the corresponding fixed zenith angle.

groundz (***Input, double precision***) Ground altitude (in m).

olxs (***Output, double precision, array(`nobslev`)***) Slant atmospheric depths (in $\mathrm{g/cm^2}$) of the corresponding observing levels.

opencrofile

FORTRAN
```
call opencrofile(wdir, filename, header1,
                 logbase, vrb, channel, irc)
```
C
```
opencrofilec(&wdir, &filename, &header1,
             &logbase, &vrb, &channel, &irc);
```

Opening a CIO file for reading. This routine performs both the system open operation and file header processing and checking.

Arguments:

wdir (***Input, character string***) The name of the directory where the file is placed. It defaults to the current directory when blank.

filename (***Input, character string***) The name of the file to open.

header1 (***Input, integer***) Integer switch to select reading (greater than or equal to 0) or skipping (less than 0) the first part of the header.

logbase (***Input, integer***) Variable to control the logarithmically scaled fields of the file records. If **logbase** is less than 2, then the returned logarithms will be natural logarithms. Otherwise base **logbase** will be returned (decimal ones if $\mathbf{logbase} = 10$).

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

channel (***Output, integer***) File identification. This variable should not be changed by the calling program. It must be used as a parameter of the reading and closing routines in order to specify the corresponding file.

irc (***Output, integer***) Return code. 0 means successful return. 1 means successful return obtained with a file that was written with a previous AIRES version. 10 means that the file could be opened normally, but that it seems not to be a valid AIRES compressed data file, or is a corrupted file; 12 invalid file header; 14 not enough size in some of the internal arrays; 16 format incompatibilities. 20: too many compressed files already opened. $300 < \mathbf{irc} < 400$ indicates a version incompatibility (when processing files written with other AIRES version) or invalid version field (corrupt header). Any other value indicates an opening / header-reading error (**irc** equals the system return code plus 10000).

raninit

FORTRAN `call raninit(seed)`
C `raninit(&seed);`

Initialization of the uniform pseudo-random number generator. This routine *must* be called before the first invocation of **grandom**, **urandom**, or **urandomt**.

Arguments:

seed (***Input, double precision***) Seed to initialize the random series. If **seed** does not belong to the interval $(0, 1)$, then the seed actually used for initialization is internally generated using the elementary generator **clockrandom**.

regetcrorecord

FORTRAN
```
okflag = regetcrorecord(channel, intfields,
                        realfields, altrec, vrb,
                        irc)
```
C
```
okflag = regetcrorecord(&channel, &intfields[1],
                        &realfields[1], &altrec,
                        &vrb, &irc);
```

Re-reading the current record. The input and output parameters of this routine are equivalent to the respective arguments of routine **getcrorecord**. The difference between this routine and the mentioned one is that **regetcrorecord** re-scans the last read record instead of advancing across the input file. **regetcrorecord** is thought to be used jointly with **getcrorectype**, **crorecfind** and other related procedures.

Arguments:

channel (***Input, integer***) Variable that uniquely identifies the I/O channel assigned to the corresponding file. This variable must be already set by means of routine **opencrofile**.

intfields (***Output, integer, array(*)***) Integer fields of the record. For a complete description of this argument see routine **getcrorecord**

realfields (***Output, double precision, array(*)***) Real fields of the record. For a complete description of this argument see routine **getcrorecord**

altrec (***Output, logical***) Alternative/default record type label. See **getcrorecord**

vrb (***Input, integer***) Verbosity control. If **vrb** is zero or negative then no error or informative messages are printed; error conditions are communicated to the calling program via the return code. If **vrb** is positive error messages will be printed: $\mathbf{vrb} = 1$ means that messages will be printed even with successful operations. $\mathbf{vrb} = 2, 3$ means that only error messages will be printed. $\mathbf{vrb} > 3$ is similar to $\mathbf{vrb} = 3$, but with the additional action of stopping the program if a fatal error takes place.

irc (***Output, integer***) Return code. For a complete description of this argument see routine **getcrorecord**

Returned value: (***Logical***) True if a record was successfully re-read. False otherwise (EOF or I/O error).

sp1stint

FORTRAN `call sp1stint(csys, x1, y1, z1, irc)`
C `sp1stint(&csys, &x1, &y1, &z1, &irc);`

Setting manually the position of the first interaction. When using special primary particles processed by external modules which may inject more that a single primary, AIRES cannot determine automatically the point where the first interaction takes place, and will take it as equal to the injection point unless it is set explicitly using **sp1stint**. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

csys (***Input, integer***) Parameter labeling the coordinate system used. $\mathbf{csys} = 0$ selects the AIRES coordinate system. $\mathbf{csys} = 1$ selects the shower axis-injection point system defined in section 3.5.

x1, y1, z1 (***Input, double precision***) Coordinates of the first interaction point with respect to the chosen coordinate system (in meters).

irc (***Output, integer***) Return code. 0 means normal return.

spaddnull

FORTRAN `call spaddnull(pener, pwt, irc)`
C `spaddnull(&pener, &pwt, &irc);`

Adding a *null* (unphysical) particle to the list of primaries to be passed from the external module to the main simulation program. This "particle" will not be propagated, but its energy will be added to the unphysical particle counter included in the shower energy balance. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

pener (***Input, double precision***) Energy (GeV).

pwt (***Input, double precision***) Null particle weight. Must be equal or greater than one.

irc (***Output, integer***) Return code. 0 means normal return.

spaddp0

FORTRAN
```
call spaddp0(pcode, pener, csys, ux, uy, uz,
                    pwt, irc)
```
C
```
spaddp0(&pcode, &pener, &csys, &ux, &uy,
        &uz, &pwt, &irc);
```

Adding a primary particle to the list of primaries to be passed from the external module to the main simulation program. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

pcode (***Input, integer***) Particle code, accordingly with the AIRES coding system described in section 2.2.1 (page 17).

pener (***Input, double precision***) Kinetic energy (GeV).

csys (***Input, integer***) Parameter labeling the coordinate system used. $\mathbf{csys} = 0$ selects the AIRES coordinate system. $\mathbf{csys} = 1$ selects the shower axis-injection point system defined in section 3.5.

ux, uy, uz (***Input, double precision***) Direction of motion with respect to the chosen coordinate system. The vector $(\mathbf{ux}, \mathbf{uy}, \mathbf{uz})$ does not need to be normalized.

pwt (***Input, double precision***) Particle weight. Must be equal or greater than one.

irc (***Output, integer***) Return code; can be one of the following:

- 0 The particle was successfully added.
- 8 Negative kinetic energy.
- 9 Particle weight less than 1.
- 10 The direction of motion is a null vector.
- 11 Invalid coordinate system specification.

spaddpn

FORTRAN
```
call spaddpn(n, pcode, pener, csys, ldu,
                    uxyz, pwt, irc)
```
C
```
spaddpn(&n, &pcode, &pener, &csys, &ldu,
          &uxyz[1][1], &pwt, &irc);
```

Adding a set of **n** primary particles to the list of primaries to be passed from the external module to the main simulation program. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

n (***Input, integer***) The number of particles to add to the list.

pcode (***Input, integer, array(n)***) Particle codes, accordingly with the AIRES coding system described in section 2.2.1 (page 17).

pener (***Input, double precision, array(n)***) Kinetic energies (GeV).

csys (***Input, integer***) Parameter labeling the coordinate system used. $\mathbf{csys} = 0$ selects the AIRES coordinate system. $\mathbf{csys} = 1$ selects the shower axis-injection point system defined in section 3.5.

ldu (***Input, integer***) Leading dimension of array **uxyz**; must be equal or greater than 3.

uxyz (***Input, double precision, array(ldu, n)***[5]) Directions of motion with respect to the chosen coordinate system. The vectors $(\mathbf{uxyz}(1,i), \mathbf{uxyz}(2,i), \mathbf{uxyz}(\mathbf{3},\mathbf{i}))$, $i = 1, \ldots, n$, do not need to be normalized.

pwt (***Input, double precision, array(n)***) Particle weights. The weights must be equal or greater than one.

irc (***Output, integer***) Return code. 0 means normal return.

[5] If **uxyz** is defined in a C environment, then its two dimensions should be swapped, i.e., `double uxyz[n][ldu]`.

speiend

FORTRAN `call speiend(retcode)`
C `speiend(&retcode);`

Closing the interface for the special primary particle external process. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

retcode (***Input, integer***) Return code to pass to the main simulation program. $\mathbf{retcode} = 0$ means normal return. If **retcode** is not zero, a message will be printed and saved in the log file (extension **.lgf**). $0 < |\mathbf{retcode}| < 10$, $10 \leq |\mathbf{retcode}| < 20$, $20 \leq |\mathbf{retcode}| < 30$, and $|\mathbf{retcode}| \geq 30$ correspond, respectively, to information, warning, error and fatal messages.

speigetmodname

FORTRAN `call speigetmodname(mn, mnlen, mnfull, mnfullen)`
C `speigetmodnamec(&mn, &mnlen, &mnfull, &mnfullen);`

Getting the name of the module invoked by the simulation program, that is, the one specified in the definition of the corresponding special particle. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

mn (***Output, string***) Name of external module. The calling program must ensure there is enough space to store the string.

mnlen (***Output, integer***) Length of external module name.

mnfull (***Output, string***) Full name of external module (Will be different of **mn** if the module was placed within one of the directories specified with the **InputPath** directive. The calling program must ensure there is enough space to store the string.

mnfullen (***Output, integer***) Length of full external module name.

speigetpars

FORTRAN `call speigetpars(parstring, pstrlen)`
C `speigetparsc(&parstring, &pstrlen);`

Getting the parameter string specified in the IDL instruction that defines the corresponding special particle. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

parstring (***Output, string***) Parameter string. The calling program must ensure there is enough space to store the string.

pstrlen (***Output, integer***) Length of parameter string. Zero if there are no parameters.

speimv

FORTRAN `call speimv(mvnew, mvold)`
C `speimv(&mvnew, &mvold);`

Setting and/or getting the external macro version. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

mvnew (***Input, integer***) Macro version number. Must be an integer in the range $[1, 759375]$. If **mvnew** is zero, then the macro version is not set.

mvold (***Output, integer***) Macro version number effective at the moment of invoking the routine. This variable will be set to zero in the first call to **speimv**.

spinjpoint

FORTRAN
```
call spinjpoint(csys, x0, y0, z0, tsw, t0beta,
                irc)
```
C
```
spinjpoint(&csys, &x0, &y0, &z0, &tsw,
           &t0beta, &irc);
```

Setting the current injection point for primary particles. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

csys (***Input, integer***) Parameter labeling the coordinate system used. $\mathbf{csys} = 0$ selects the AIRES coordinate system. $\mathbf{csys} = 1$ selects the shower axis-injection point system defined in section 3.5.

x0, y0, z0 (***Input, double precision***) Coordinates of the injection point with respect to the chosen coordinate system (in meters).

tsw (***Input, integer***) Injection time switch. If **tsw** is zero then **t0beta** is an absolute injection time; if **tsw** is 1, then the injection time is set as the time employed by a particle whose speed is $\mathbf{t0beta} \times c$ to go from the original injection point to the intersection point of the shower axis with the plane orthogonal to that axis and containing the point $(\mathbf{x0}, \mathbf{y0}, \mathbf{z0})$.

t0beta (***Input, double precision***) The meaning of this argument depends on the current value of **tsw**. It can be the absolute injection time (ns) (time at original injection is taken as zero); or the speed of a particle divided by c.

irc (***Output, integer***) Return code. 0 means normal return.

speistart

FORTRAN
```
call speistart(showerno, primener, injpos,
                    xvinj, zground, xvground,
                    dgroundinj, uprim)
```
C
```
speistart(&showerno, &primener, &injpos[1],
          &xvinj, &zground, &xvground,
          &dgroundinj, &uprim[1]);
```

Starting the interface for the special primary particle external process. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

showerno (***Output, integer***) Current shower number.

primener (***Output, double precision***) Primary energy (GeV).

injpos (***Output, double precision, array(3)***) Position of the initial injection point with respect to the AIRES coordinate system (in meters).

xvinj (***Output, double precision***) Vertical atmospheric depth of the injection point (in $\mathrm{g/cm^2}$).

zground (***Output, double precision***) Altitude og ground level (in m.a.s.l.).

xvground (***Output, double precision***) Vertical atmospheric depth of the ground surface (in $\mathrm{g/cm^2}$).

dgroundinj (***Output, double precision***) Distance from the injection point to the intersection between the shower axis and the ground surface (in meters).

uprim (***Output, double precision, array(3)***) Unitary vector in the direction of the straight line going from the injection point towards the intersection between the shower axis and the ground plane.

speitask

FORTRAN `call speitask(taskn, tasklen, tver)`
C `speitaskc(&taskn, &tasklen, &tver);`

Getting the current task name and version. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

taskn (***Output, string***) Task name. The calling program must ensure there is enough space to store the string.

tasklen (***Output, integer***) Length of task name.

tver (***Output, integer***) Task name version.

spnshowers

FORTRAN `call spnshowers(totsh, firstsh, lastsh)`
C `spnshowers(&totsh, &firstsh, &lastsh);`

Getting the current values of the first and last shower, and total number of showers. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

totsh (***Output, integer***) Total number of showers for the current task.

firstsh (***Output, integer***) Number of first shower.

lasttsh (***Output, integer***) Number of last shower.

sprimname

FORTRAN `call sprimname(pname, pnamelen)`
C `sprimnamec(&pname, &pnamelen);`

Getting the name of the special primary particle specified in the corresponding IDL instruction. This routine should be used only within modules designed to process special primaries, and following the guidelines of section 3.5 (page 63).

Arguments:

pname (***Output, string***) The name of the special particle. The calling program must ensure there is enough space to store the string.

pnamelen (***Output, integer***) Length of particle name.

thisairesversion

FORTRAN `iavers = thisairesversion()`
C `iavers = thisairesversion();`

Returning the current version of the AIRES library.

Returned value: (***Integer***) The corresponding version in integer format (for example the number 01040200 for version 1.4.2, 01040201 for version 1.4.2a, etc.).

urandom

FORTRAN `r = urandom()`
C `r = urandom();`

This function invokes the AIRES random number generator and returns a pseudo-random number uniformly distributed in the interval $[0, 1)$. It is necessary to initialize the random series calling **raninit** before using this function.

Returned value: (***Double precision***) The uniform pseudo-random number.

urandomt

FORTRAN `r = urandomt(threshold)`
C `r = urandomt(threshold);`

This function invokes the AIRES random number generator and returns a pseudo-random number uniformly distributed in the interval $[t, 1)$, where t is a specified threshold ($0 \leq t < 1$). It is necessary to initialize the random series calling **raninit** before using this function.

Arguments:

threshold (***Input, double precision***) The threshold t.

Returned value: (***Double precision***) The uniform pseudo-random number.

xslant

```
FORTRAN   X = xslant(Xvert, Xv0, cozenith,
                     zground)
C         X = xslant(&Xvert, &Xv0, &cozenith,
                     &zground);
```

Converting vertical atmospheric depths into slant atmospheric depths. This routine evaluates the slanted path (in $\mathrm{g/cm^2}$) of equation (2.8), starting (ending) at the point whose *vertical* depth is **Xvert** (**Xv0**). The inclination of the integration path is controlled by parameters **cozenith** (cosine of the zenith angle) and **zground** (altitude, in meters, of the intersection between the oblique axis and the z-axis), as illustrated in figure 2.1 (page 9).

Arguments:

Xvert (***Input, double precision***) Vertical atmospheric depth (in $\mathrm{g/cm^2}$) of the point marking the beginning of the integration path. Must be positive.

Xv0 (***Input, double precision***) Vertical atmospheric depth (in $\mathrm{g/cm^2}$) of the point marking the end of the integration path. If **Xv0** is zero, then the end of the integration path is the top of the atmosphere. If **Xv0** corresponds to a point located below the point corresponding to **Xvert** ($\mathbf{Xv0} > \mathbf{Xvert}$), then the returned slant depth will be negative.

cozenith (***Input, double precision***) Cosine of the zenith angle Θ (see figure 2.1) corresponding to the integration line. Must be in the range $(0, 1]$.

zground (***Input, double precision***) z-coordinate (in meters) of the intersection point between the oblique axis and the z-axis, which is normally coincident with the "ground altitude".

Returned value: (***Double precision***) The slant atmospheric depth in $\mathrm{g/cm^2}$; or zero in case of error or invalid argument.

zfromdepth

FORTRAN `z = zfromdepth(Xvert, atlayer)`
C `z = zfromdepth(&Xvert, &atlayer);`

Vertical altitude from atmospheric depth. Multilayer atmospheric model..

Arguments:

Xvert (***Input, double precision***) Vertical atmospheric depth (in g/cm^2) of the corresponding point. Must be positive.

atlayer (***Output, integer***) Atmospheric layer corresponding to the depth **Xvert**. This parameter depends on the selected atmospheric model.

Returned value: (***Double precision***) The vertical altitude in m.a.s.l.

References

[1] A. M. Hillas, *Nucl. Phys. B (Proc. Suppl.),* **52B**, 29 (1997); A. M. Hillas, *Proc. 19th ICRC (La Jolla),* **1**, 155 (1985).

[2] S. J. Sciutto, *AIRES: A minimum document,* Auger technical note GAP-97-029 (1997).

[3] M. T. Dova and S. J. Sciutto, *Air Shower Simulations: Comparison Between AIRES and MOCCA,* Auger technical note GAP-97-053 (1997).

[4] A. M. Hillas, *Proc. of the Paris Workshop on Cascade simulations,* J. Linsley and A. M. Hillas (eds.), p 39 (1981).

[5] M. Kobal, A. Filipčič and D. Zavrtanik, Auger technical notes GAP-98-001 and GAP-98-058 (1998).

[6] T. Pierog, Iu. Karpenko, J.M. Katzy, E. Yatsenko, K. Werner, *Phys. Rev. C,* **92**, 034906 (2015).

[7] S. S. Ostapchenko, *Phys. Rev. D,* **83**, 014018 (2011).

[8] S. Ostapchenko, *Nucl. Phys. B (Proc. Suppl.),* **151**, 143 (2006).

[9] N. N. Kalmykov and S. S. Ostapchenko, *Yad. Fiz.,* **56**, 105 (1993); *Phys. At. Nucl.,* **56**, (3) 346 (1993); N. N. Kalmykov, S. S. Ostapchenko and A. I. Pavlov, *Bull. Russ. Acad. Sci. (Physics),* **58**, 1966 (1994).

[10] Eun-Joo Ahn *et al., Phys. Rev. D,* **80**, 094003 (2009); F. Riehn *et al., Proc. 35th Int. Cosmic Ray Conf. (Bexco, Busan, Korea),* , cont. 301 (2017).

[11] Eun-Joo Ahn *et al., Phys. Rev. D,* **80**, 094003 (2009); F. Riehn *et al., Proc. 35th Int. Cosmic Ray Conf. (The Hague, The Netherlands),* , cont. 1313 (2015).

[12] R. Engel, T. K. Gaisser, T. Stanev, *Proc. 26th ICRC (Utah),* **1**, 415 (1999).

[13] R. T. Fletcher, T. K. Gaisser, P. Lipari and T. Stanev, *Phys. Rev. D,* **50**, 5710 (1994); J. Engel, T. K. Gaisser, P. Lipari and T. Stanev, *Phys. Rev. D,* **46**, 5013 (1992).

[14] H. Moritz, *J. of Geodesy,* **74**, 128 (2000).

[15] The data, software and documentation related with the International Geomagnetic Reference Field are distributed by the *National Geophysical Data Center,* Boulder (CO), USA, and can be obtained electronically at the following Web address: **`www.ngdc.noaa.gov/IAGA/vmod`**.

[16] NETLIB is a public collection of mathematical software, papers, and databases, that can be accessed through Internet, at the World Wide Web address **`www.netlib.org`**.

[17] CERN Program library Long Writeup **Q121** (1995).

[18] S. J. Sciutto, *AIRES users guide and reference manual ,* version 1.4.2, Auger technical note GAP-98-032 (1998).

[19] National Aerospace Administration (NASA), National Oceanic and Atmospheric Administration (NOAA) and US Air Force, *US standard atmosphere 1976,* NASA technical report NASA-TM-X-74335, NOAA technical report NOAA-S/T-76-1562 (1976).

[20] R. C. Weast (editor), *CRC Handbook of Chemistry and Physics, 61st edition,* pp F206 – F213, CRC Press, Boca Raton (FL, USA) (1981).

[21] B. Rossi, *High-energy particles,* Prentice-Hall, New Jersey (USA) (1956).

[22] We were not able to find official references related with Linsley's standard atmosphere model. References [1, 36] contain information about parameterization data.

[23] A. Cillis and S. J. Sciutto, *J. Phys. G,* **26**, 309-321 (2000).

[24] A. Cillis and S. J. Sciutto, *Phys. Rev. D,* **64**, 013010 (2001).

[25] A. B. Migdal, *Phys. Rev.,* **103**, 1811 (1956).

[26] A. Cillis, C. A. Garcıa Canal, H. Fanchiotti and S. J. Sciutto, *Phys. Rev. D,* **59**, 113012 (1999).

[27] D. Heck, private communication.

[28] T. K. Gaisser, *Cosmic Rays and Particle Physics,* Cambridge University Press, Cambridge (1992).

[29] S. J. Sciutto, in preparation.

[30] L. D. Landau and I. Ya. Pomeranchuk, *Dokl. Akad. Nauk SSSR,* **92**, 535, 735 (1953).

[31] S. Klein, preprint hep-ph/9820442 (1998).

[32] I. Vattulainen, T. Ala-Nissila, K. Kankaala, *Phys. Rev. Lett.* **73**, 2513 (1994).

[33] S. J. Sciutto, in preparation.

[34] V. S. Berezinskiĭ, *et. al.,* V. L. Ginzburg (editor), *Astrophysics of cosmic rays,* North-Holland (1990).

[35] T. K. Gaisser and A. M Hillas, *Proc. 15th ICRC (Plovdiv),* **8**, 353 (1977).

[36] D. Heck, J. Knapp, J.N. Capdevielle, G. Schatz, and T. Thouw, *Forschungszentrum Karlsruhe,* Report FZKA 6019 (1998).

[37] P. Billoir, private communication.

[38] Particle Data Group, M. Tanabashi *et al.* (Particle Data Group), *Phys. Rev. D,* **98**, 030001 (2018).

[39] CERN Program library Long Writeup **W5013** (1994).

[40] Particle Data Group, D. E. Groom *et. al., The European Physical Journal,* **C15**, 1 (2000); website: **`www-pdg.lbl.gov`**.

[41] J. C. Moreno, S. J. Sciutto, *Eur. Phys. J. Plus,* **128**, 104 (2013).

[42] J. Alvarez-Muñiz, W. R. Carvalho, A. Romero-Wolf, M. Tueros, E. Zas, *Phys. Rev. D,* **86**, 123007 (2012).

Index

Page numbers in **boldface** represent the definition or the main source of information about whatever is being indexed.