

User's guide for the APOLLO procedure (version 1.1)

Arnau Folch (afolch@ov.ingv.it)
Antonio Costa (costa@ov.ingv.it)
Giovanni Macedonio (macedon@ov.ingv.it)

Istituto Nazionale di Geofisica e Vulcanologia
Sezione "Osservatorio Vesuviano"
Via Diocleziano 326 I-80124 Napoli, Italy

January 2008

Contents

1	Foreground	3
1.1	About this manual	3
1.2	Introduction	3
1.3	Overview of the APOLLO procedure	4
1.4	Download and installation	5
2	Generation of a meteorological database	7
2.1	Overview	7
2.2	Description of programs	8
2.2.1	The program GEOINP	8
2.2.2	The program MESOINP	9
2.2.3	The program CALMETINP	10
2.2.4	The program CALMET	10
2.2.5	The program BUILDDBS	10
2.2.6	The program POSTPDBS	11
2.3	The database input file	11
2.4	The database script files	14
3	Run generation	15
3.1	Overview	15
3.2	Generation of a granulometry file. The program SETGRANUM	15
3.3	Generation of a source file. The program SETSRC	16
3.4	Fallout models	16
3.4.1	HAZMAP model	16
3.4.2	FALL3D model	18
3.4.3	TEPHRA model	20
3.5	The run input file	22
3.6	The run script files	28
4	Postprocess of models	29
4.1	Overview	29
4.2	The program MODELPOSTP	29
5	The library LIBAPOLLO	30
5.1	Routines to read an input control file	30
5.2	Routines to read a database	31
5.3	Routines to read a source file	34
5.4	Routines to read a granulometry file	35
5.5	Routines to read a wind profile file	36
5.6	Routines to output model results	36
6	File formats	39
6.1	The terrain file format	39
6.2	The wind profile file format	39
6.3	The granulometry file format	40
6.4	The source file format	40
6.5	The model output file format	41
6.6	The GRD file format	42
6.7	The symbols file format	43
7	The default APOLLO tree	45
8	Application example	48
9	References	51

1 Foreground

1.1 About this manual

This manual has been prepared by Arnau Folch and Antonio Costa. It gives general instructions to install and run the APOLLO procedure, version 1.1. The software is freely distributed for non-commercial purposes. The authors decline any responsibility for any error or incorrect use. Please note that this version of APOLLO is a beta version still under test. If you find any bug please report it to us.

1.2 Introduction

Explosive volcanic eruptions can eject into the atmosphere large amounts of blocks, lapilli and ash during sustained periods of time. These products, globally known as tephra, represent a serious threat for communities settled around active volcanoes. It is estimated that half a billion people live nowadays close to active volcanoes (Small and Naumann , 2001). Several tens of cities and urban areas near volcanoes exceed one million inhabitants including, just to mention some relevant examples, Mexico City, Tokyo, Manila, Quito, Seattle or Naples (Chester et al. , 2001). Approximately 500 airports lie within 100 km of volcanoes that have erupted during the last hundred years, and tens of thousands of passengers fly over volcanically active regions such as the North Pacific, which has more than 100 active volcanoes and four to five ash-producing eruptions per year (Casadevall , 1993). These data stress the potential socio-economic impacts of volcanoes in general, of ash fallout in particular, and highlight the relevance of an adequate hazard assessment and risk mitigation policies. On the other hand, reliable short-term forecasts represent a valuable aid for scientists and civil authorities to mitigate the effects of fallout on the surrounding areas during an episode of crisis. In such a context it is essential to have accurate models for volcanic fallout.

An increasing number of models to predict ash transport and/or the characteristics of the resulting fallout deposits have been developed during the last decades. Simplest models are obviously less accurate but have lower computational requirements and hence are especially suitable to tackle inverse problems and/or to produce immediate gross predictions. In contrast, complex models are more accurate but, in general, require more inputs (not always available), set up times, pre and postprocess data treatment (*i.e.* possible involuntary manipulation errors), computational requirements and user expertise. All these factors may preclude the efficiency of such models during an episode of pre-eruptive crisis (or, even worst, during the course of an eruption) because they may delay the production and delivery of short-term forecasts to the decision-making authorities. An important challenge for the modelling community is to overcome these limitations in order to advance towards a simultaneously efficient and accurate performance of models.

The goal of the APOLLO procedure is to facilitate the execution of fallout models by means of an automatic acquisition and manipulation of input data, a subsequent automation of runs and a final shared postprocess analysis. The idea is to increase performance, eliminate involuntary human manipulation errors, speed up computing times and anticipate the scientific response during emergencies. Moreover, another no trivial advantage is that models share inputs and postprocess treatment through the production of maps written in portable formats which allow for immediate comparison among outcomes from different models.

1.3 Overview of the APOLLO procedure

APOLLO (acronym for *Automatic Procedure to mOdeL voLcanic ash dispersiOn*) is a platform-independent procedure designed to facilitate the execution and subsequent interpretation of volcanic ash transport and fallout models. The APOLLO procedure is built on a series of open-source programs that perform different tasks, generate input data needed by models and do simple postprocessing. Three open-source fallout models, HAZMAP , FALL3D (both serial and parallel versions) and TEPHRA , are included in this version of APOLLO. However, the user is not constrained to these models but can, alternatively, add other model(s) with minor modifications on the source codes. To this purpose, APOLLO contains a library (named LIBAPOLLO) that acts as an interface between programs/models and input data files. Data from files and databases generated by different programs included in the APOLLO procedure can be read directly through simple LibApollo routine calls (using either FORTRAN or C-C++) without having a detailed knowledge of the file/database format.

The APOLLO procedure generates all the data needed by models, including a terrain and a meteorological database, the definition of the source term and the granulometric distribution. A meteorological database for a particular area contains short-term predictions, typically up to few days, for meteorological variables (*e.g.* wind field, temperature, turbulence related variables, *etc.*) defined at the nodes of a 3D structured grid. The meteorological database(s) is(are) absolutely independent from models and can be updated automatically, typically every day as new meteorological prognostics are available. A run can start automatically after the construction of a meteorological database or at any user defined time (a run is mainly an scenario; it may content several simulations from different fallout models). Whenever a fallout model runs it simply reads the required meteorological (and, if necessary terrain) data from the database as well as the files that define the source and the granulometric data. Clearly, the kind of data to read varies from model to model (a model is not constrained to use the entire contents of the database). For example, if a model assumes that the wind field is horizontally uniform it is sufficient to use a selected value from each vertical layer of the database, for instance the average or a manually specified profile. The gathering of data from a database is, consequently, a model dependent step and must be implemented *ad hoc* for each particular model. After a run, the last step of the procedure is to postprocess the outcomes of models in order to draw maps with pre-defined physical quantities. All models can share the same postprocess treatment, so that if two or more different models output the same quantity (*e.g.* ground deposit thickness) their respective maps are directly comparable.

The user introduces inputs by means of short ASCII control files. There are two kinds of input files: meteorological database input files and run input files. The formers control the parameters that define a meteorological database (each database must have its own input file). The latter control the parameters that define a run (each run must have its own input file). Input control files can be modified at any time, for example, to incorporate data acquired during an on-going eruption (*e.g.* measurements of granulometry or column height, estimations of the mass flow rate, *etc.*). If the control input files remain unmodified models run periodically with the same eruptive parameters but using updated meteorological predictions. The latter scenario could be characteristic of a pre-eruptive crisis period, during which the eruptive parameters (*e.g.* mass flow rate, granulometry, *etc.*) must necessarily be guessed based on the experience from previous events.

Some advantages of the APOLLO procedure are:

1. Modularity. Each program of the procedure performs a specific task and runs independently from the rest. It gives large flexibility and facilitates future modifications or addition of new functionalities.
2. Flexibility. There is an absolute flexibility concerning the quantity of meteorological data-

bases and number of runs. For instance, different databases for different regions can coexist and be updated with different periodicity (*e.g.* every 6 hours, daily, *etc.*). It allows, for instance, to automate forecasts for several volcanoes or volcanic areas simultaneously. On the other hand, there is no limit on the number of runs for a specific location (several runs can use the same meteorological database). Thus, for example, one could consider different runs starting at the same time instant (*e.g.* to model an event supposed to start after 24 hours but considering different scenarios characterized by different mass flow rates or column heights), different runs starting at different time instants (*e.g.* to model a single scenario supposed to start after 24, 48 or 72 hours), or both.

3. Automatization. The scripts that control the flow of the procedure can be launched periodically without user intervention. It speeds up the production of results and precludes from user manipulation errors.
4. Data sharing. All the models run using the same input data and can share also the same postprocess. It ensures that outcomes (maps) from different models in the same run are directly comparable.
5. Model/data independency. Models and data interface through a library (the LIBAPOLLO). It guarantees that future changes in the formats of files will not affect models and vice-versa.
6. Open source. All the programs of the procedure are distributed freely for non-commercial purposes. A user can add also new models or functionalities and optionally make them accessible.

The APOLLO procedure flows by means of simple scripts which can be launched either manually or automatically with a user defined periodicity. The script files, located by default in the folder `scripts` (see section 7 for details on the APOLLO tree structure), define the name and location of the files and call the programs in the appropriate order. Any program is called passing the path (including name) of the required input and output files as an argument. Output files are created with the name and location specified by the argument.

1.4 Download and installation

Requirements: A FORTRAN and a C compiler. In addition, MPI (version 2.0) is also necessary to run the parallel version of the FALL3D model.

- On a Unix/Linux/Mac X operating system:

1. Download the compressed file `apollo-1.1.tar.gz` from the APOLLO site.
2. Decompress (type “`gunzip apollo-1.1.tar.gz`”) and then untar (type “`tar -xvf apollo-1.1.tar`”) to obtain the directory *apollo*.
3. Go to the folder *Scripts/Master* and run the script “APOLLO-Install” specifying which compiler you want to use. This script does successive calls to the *Makefiles* of the different programs and models. On a Unix/Linux/Mac X OS the “APOLLO-Install” script assumes that either INTEL ifort or GNU gfortran as well as GNU gcc are available (to compile using other compilers you will need to change the affected *Makefiles* and launch them manually). Type:
 - “APOLLO-Install fcompiler_ifort” (for INTER ifort compiler).
 - “APOLLO-Install fcompiler_gfortran” (for GNU gfortran compiler).

- On a Windows operating system:
 1. Download the compressed file `apollo-1.1.tar.gz` from the APOLLO site.
 2. Decompress to obtain the directory *apollo*.
 3. Compile the different programs and models using your favourite FORTRAN and C compilers. No automatic installation is provided for Windows OS.

2 Generation of a meteorological database

2.1 Overview

Fallout models need meteorological data as input (simplest models may require only a vertical wind field profile, more elaborated models normally need 3D time-dependent wind fields as well as other micrometeorological variables). The APOLLO procedure builds a meteorological database for each region of interest. A database contains the topography of the region and relevant prognostic meteorological variables: wind field, air temperature, velocity scales, Monin-Obukhov length, and mixing height. Discrete prognostic values for these variables at different time instants are stored at the nodes of a 3D grid. This grid is regular (equally spaced) along the horizontal but can have an arbitrary vertical layering. It allows to define meteorological data grids finer within the Atmospheric Boundary Layer (ABL), where gradients of meteorological variables are larger, and coarser at higher atmospheric levels. Models are not constrained to run on the same grid where meteorological data are stored. Models and database interface through the library LIBAPOLLO, which contains routines that extract values from a database either at a single point or at a horizontal layer.

There are two ways to construct a database. The simplest one requires a vertical profile of temperature and wind speed (normally obtained from a vertical sounding). The second option, more elaborated, is based on the program CALMET (Scire et al., 2000), an open source meteorological processor developed and maintained by scientists of the US Atmospheric Studies Group (ASG) which includes a diagnostic wind field generator. APOLLO takes advantage of this functionality to obtain time-dependent wind and temperature fields as well as other micrometeorological variables. Assimilating terrain information and an initial guess wind field on a coarse mesh, CALMET (version 6.2) computes a zero-divergence wind field and other diagnostic variables on a finer grid using a terrain following coordinate system. CALMET gives the option to use a gridded wind as furnished by a prognostic meteorological model as the initial guess wind field. Note that prognostic meteorological models run on significantly larger horizontal grid spacing ($\approx 100\text{km}$ for synoptic models and $\approx 10\text{km}$ for mesoscale models) and different vertical layering than those of CALMET. Consequently, CALMET interpolates the guess field from the grid of the prognostic model to its own grid.

Figure 1 illustrates the flow to create/update a meteorological database. When using the CALMET option, the steps include:

1. Download the files that contain the (mesoscale) meteorological prognostics. This step is not done by the APOLLO procedure. The user is responsible for periodic (daily) download and storage of data (normally a set of files written in GRIB or NetCDF formats that contain mesoscale prognostics every 3, 6 or 12 hours). The choice of a specific mesoscale model may depend upon several factors, but the spatial coverage of the model and the facilities to get access to data are, obviously, two determinant factors.
2. Run the CALMET processor. In order to facilitate the execution of CALMET, APOLLO contains a set of programs (GEOINP, MESOINP, and CALMETINP, see section 2.2 for description) that act as interfaces gathering data and creating the CALMET input files. All these programs have to run each time meteorological data is updated.
3. Run the BUILDDBS program to construct the meteorological database either from a CALMET output or from a vertical profile (in the latter case steps 1 and 2 are unnecessary).
4. Optionally, run also the program POSTPDBS that allows a simple visualization of meteorological data.

Steps 2 to 4 can be performed automatically by means of the script APOLLO-Build-Dbs-DbsName (see section 2.4).

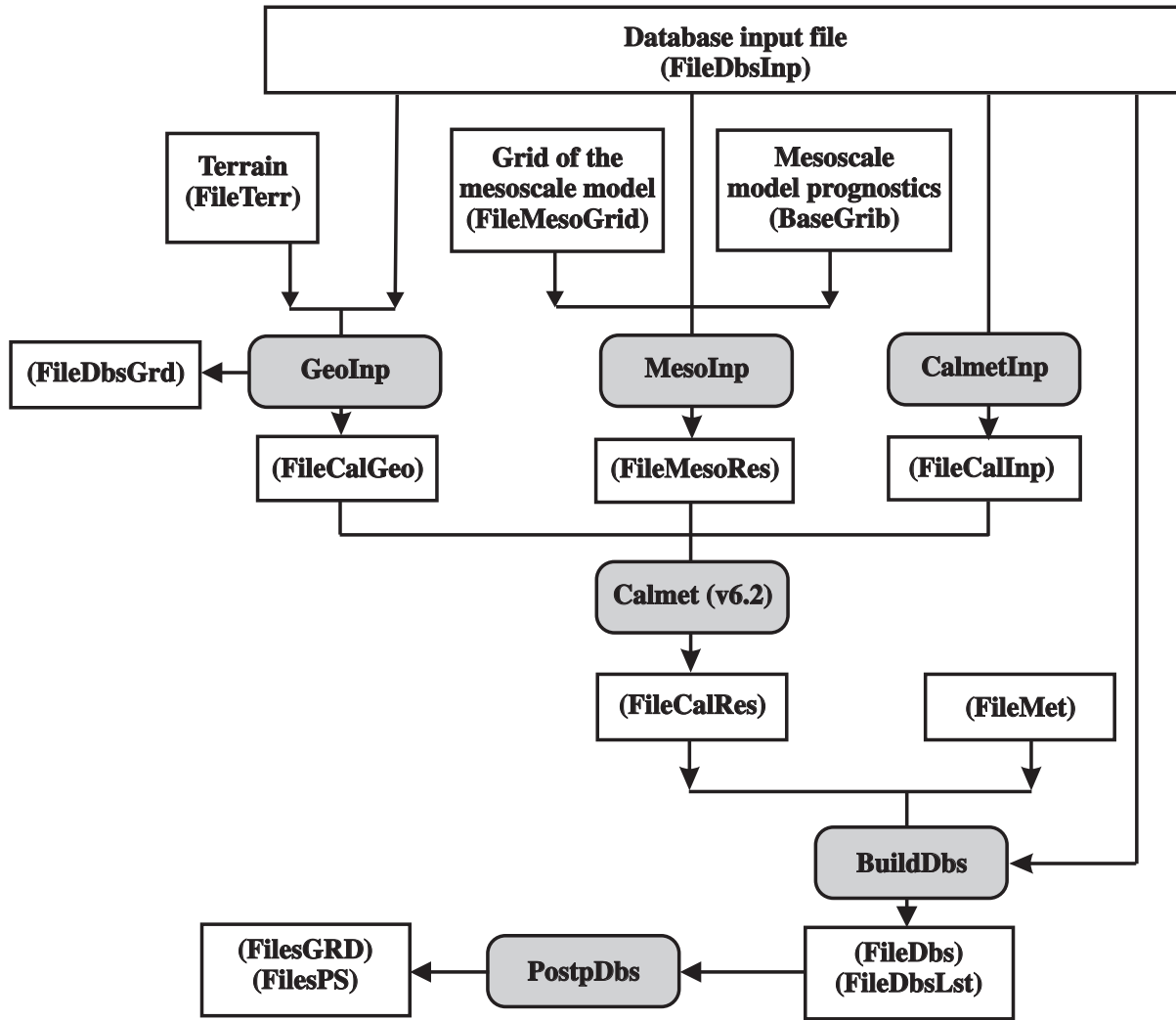


Figure 1: Summary of the database construction/update flow.

2.2 Description of programs

2.2.1 The program GEOINP

DESCRIPTION: CALMET requires an input file with 2D geophysical data at ground level. Data in this file include terrain elevation, land use type, surface parameters (surface roughness, albedo, Bowen ratio, soil heat flux and leaf area index) and anthropogenic heat flux. The program GEOINP (alias for GEOphysical INPut generator) extracts data from a regional terrain file (**FileTerr** in Fig.1), interpolates the geophysical parameters needed by CALMET from the terrain file to the ground nodes of the database (of the CALMET grid) and, finally, writes these data in a CALMET readable format (**FileCalGeo** in Fig.1). A terrain file (see section 6.1) is a free format ASCII file that contains data at regular spaced points. Terrain data files should ideally cover several hundreds of kilometres around the volcano or volcanic area of interest and, in principle, can have an arbitrary spatial resolution. The default spacing is 1km. Note that, in general, the terrain file and the database can have different extensions and/or spatial resolutions.

The only requirement is that the domain of the database (typically of the order of 100x100 km) must lay within the bounds of the terrain file (typically of the order of 1000x1000 km).

PROGRAM CALL: (normally included in a script file): Path of the executable + 5 arguments

`“GeoInp.exe FileLog FileDbsInp FileCalGeo FileTerr FileDbsGrd”`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileDbsInp:** Path (including name) of the database input file (see section 2.3).
- **FileCalGeo:** Path (including name) of the CalGeo file. It is a GEOINP output ASCII file that is used by CALMET as an input.
- **FileTerr:** Path (including name) of the terrain file (see section 6.1).
- **FileDbsGrd:** Path (including name) of the DbsGrd file. It is a GEOINP output file written in GRD format that contains the domain (and topography) of the database. It serves just for optional visualization purposes.

2.2.2 The program MESOINP

DESCRIPTION: The purpose of the MESOINP (alias for MESOscale INPut generator) program is to read and decode the necessary GRIB-format files produced by the mesoscale meteorological models and subsequently merge them into a single ASCII file written in a CALMET readable format. The number of GRIB files required results from the ratio between the database time interval (selected by the user in the database input file) to the mesoscale model output time interval. For instance, to store data for the next 48 hours using meteorological data provided by a mesoscale model which supplies data every 6 hours it is necessary to decode up to 8 GRIB files.

PROGRAM CALL: (normally included in a script file): Path of the executable + 6 arguments

`“MesoInp.exe FileLog FileDbsInp FileMesoLst BaseGrib FileMesoGrid FileMesoRes”`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileDbsInp:** Path (including name) of the database input file (see section 2.3).
- **FileMesoLst:** Path (including name) of the MesoLst file. It is a MESOINP output ASCII file that contains run information.
- **BaseGrib:** Path (including name but not file extension .hh.grb) of the GRIB files to be decoded.
- **FileMesoGrid:** Path (including name) of the file that contains the grid of the mesoscale model.
- **FileMesoRes:** Path (including name) of the MesoRes file. It is a MESOINP output ASCII file that is used by CALMET as input.

2.2.3 The program CALMETINP

DESCRIPTION: The program CALMETINP (alias for Calmet Input file generator) writes the CALMET control file. This file contains all the information necessary to define a CALMET run.

PROGRAM CALL: (normally included in a script file): Path of the executable + 3 arguments

“CalmetInp.exe FileLog FileDbsInp FileCalInp”

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileDbsInp:** Path (including name) of the database input file (see section 2.3).
- **FileCalInp:** Path (including name) of the CalInp file. It is a CALMETINP output ASCII file that is used by CALMET as input.

2.2.4 The program CALMET

DESCRIPTION: The CALMET program (see Scire et al. (2000) for details) assimilates terrain information and an initial guess wind field on a coarse mesh to compute a zero-divergence wind field and other diagnostic variables on a finer grid using a terrain following coordinate system. For each time interval, the initial guess wind field (in our case the output of a meteorological prognostic model) is first adjusted for: i) kinematic effects of terrain (lifting and acceleration of the air flow over terrain obstacles), ii) thermodynamically generated slope flows and, iii) blocking effects, in order to obtain, after a divergence-minimisation procedure, a step 1 mass consistent wind field. After that, meteorological observations (if available at the time under consideration), can be added to the step 1 field and an objective analysis procedure gives a second intermediate field. The scheme is designed so that observations are used to correct the step 1 wind field within a user specified radius of influence, whereas it remains unchanged at regions where observations are unavailable. Finally, a new divergence minimisation procedure is applied iteratively to the step 2 field until the divergence of velocity reaches a lower bound. The final outcome of CALMET consists of values at the grid points for a zero-divergence wind field consistent with the observations (or pseudo observations) and for other micrometeorological variables like the Monin-Obukhov length, the friction velocity or the atmospheric boundary layer height. The latter ones are quantities that may be later required by some fallout models to estimate the eddy diffusivity tensor. It is important to note that the approximation of a zero-divergence wind field assumed by CALMET is fully adequate at heights lower or close to one kilometer (Dutton and Fichtl, 1969), although it is commonly extended up to few kilometres. In consequence, the CALMET output field can be used confidently just for low to medium eruptive columns.

PROGRAM CALL: (normally included in a script file): Path of the executable + 2 arguments

“Calmet62.exe FileLog FileCalInp”

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileCalInp:** Path (including name) of the CalInp file (CALMETINP output file).

2.2.5 The program BUILDDBS

DESCRIPTION: This program generates the database files using as input either a vertical profile (sounding) plus a topography file (in format GRD) or an output of the meteorological

processor CALMET (version 6.2). The latter option is preferable because CALMET generates a 3D wind field that accounts for topographic effects and determines values for micrometeorological variables in the Atmospheric Boundary Layer (ABL).

PROGRAM CALL: (normally included in a script file): Path of the executable + 7 arguments

`"BuildDbs.exe FileLog FileDbsInp FileDat FileDbs TypeData FileTop"`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileDbsInp:** Path (including name) of the database input file (see section 2.3).
- **FileDat:** Name (including path) of the meteo data file. This is either the vertical profile file (see section 6.2) or the binary output from CALMET (version 6.2) depending on the value of **TypeData**.
- **FileDbs:** Path (including name) of the Dbs file. It is a binary file created by the BUILDDBS program. Contains the meteorological database stored in a direct access binary file.
- **TypeData:** Flag to indicate the origin of meteorological data. Possibilities are PROFILE or CALMET62.
- **FileTop:** Name (including path) of the GRD topography file. Only used if **TypeData** is PROFILE.

2.2.6 The program POSTPDBS

DESCRIPTION: This program does an optional simple postprocess of a database. It plots horizontal cuts of meteorological variables (in this version only wind vector field and air temperature).

PROGRAM CALL: (normally included in a script file): Path of the executable + 4 arguments

`"Postpdb.exe FileLog FileDbsInp FileDbs BaseName"`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileDbsInp:** Path (including name) of the database input file (see section 2.3).
- **FileDbs:** Path (including name) of the Dbs file. It is a binary file created by the BUILDDBS program. Contains the meteorological database stored in a direct access binary file.
- **BaseName:** Path for the POSTPDBS program output files

2.3 The database input file

A database input file (see Table 1) is an ASCII file composed by a series of blocks and labels that define all the characteristics of a database. Labels are case sensitive and can be placed in any order within a block. Comments and extra lines can be inserted anywhere with no particular syntax. This file controls the input parameters needed by the different programs described in the previous section. There must exist a single input file for each meteorological database to build/update.

Block TIME_UTC (read by the programs: GEOINP, MESOINP, CALMETINP, and BUILDDBS). This block contains labels that define the time range of the meteorological database (in UTC time).

- **YEAR:** Simulation year.
- **MONTH:** Simulation month (1-12).
- **DAY:** Simulation day (1-31).
- **BEGIN_METEO_DATA_(HOURS_AFTER_00):** Time (in h after 0000UTC for the current day) at which meteorological data start.
- **END_METEO_DATA_(HOURS_AFTER_00):** Time (in h after 0000UTC for the current day) at which meteorological data ends. The meteo time interval should include the simulation time interval defined by the records **RUN_START_(HOURS_AFTER_00)** and **RUN_END_(HOURS_AFTER_00)** of the run input file (see section 3.5).
- **TIME_STEP_METEO_DATA_(MIN):** Time step (in min) of the meteo data.

Block DATABASE_GRID (read by programs GEOINP, CALMETINP, and BUILDDBS). This block contains labels that define the size and location of the database.

- **UTM_ZONE:** UTM zone code (1-60).
- **UTM_HEMISPHERE:** UTM hemisphere. Possibilities are N or S.
- **X_ORIGIN_(UTM_M):** x -origin of the database (bottom left corner). UTM coordinates in m .
- **Y_ORIGIN_(UTM_M):** y -origin of the database (bottom left corner). UTM coordinates in m .
- **CELL_SIZE_(KM):** Horizontal database cell size in km (*e.g.* 0.5, 1, 2, *etc.*).
- **NX:** Number of database grid cells in the x -direction (W-E direction). Coincides with the fallout models discretization.
- **NY:** Number of database grid cells in the y -direction (S-N direction). Coincides with the fallout models discretization.
- **ZLAYER_(M):** Heights (in m) of the database z -layers. If **TypeData** is **PROFILE** then **BUILDDBS** interpolates the measured values of velocity and temperature at these heights. If **TypeData** is **CALMET62** the heights represent the **CALMET** cell faces.

Block CALMET (read by programs MESOINP, CALMETINP, and BUILDDBS). This block contains labels that define some variables needed by **CALMET**.

- **MESOSCALE_MODEL:** Alias of the mesoscale model used. Possibilities are: **AMITA**, **LAMIB**, **ARPASIM**, **NOAA**, and **ECMWF**.
- **MESOSCALE_TIME_INCREMENT_(HOURS):** Time increment (in h) of the mesoscale data. This is usually 3 or 6. Normally each time interval corresponds to a GRIB file.
- **MESOSCALE_RANGE_OF_LATITUDES:** An interval of latitudes that contains the database. This is just to speed up the algorithm that searches which points of the mesoscale model grid lay within the domain of the database.
- **MESOSCALE_RANGE_OF_LONGITUDES:** An interval of longitudes that contains the database. This is just to speed up the algorithm that searches which points of the mesoscale model grid lay within the domain of the database.

Block POSTPROCESS_DBS (read by program POSTPDBS). This block contains labels that define the variables needed by the optional database postprocessor program.

- **OUTPUT_FILES_IN_GRD_FORMAT**: Possibilities are YES or NO. If YES plots maps in GRD file format.
- **OUTPUT_FILES_IN_PS_FORMAT**: Possibilities are YES or NO. If YES plots maps in PS file format.
- **POSTPROCESS_TIME_INTERVAL_(HOURS)**: Time interval (in h) of postprocess plots, starting from the database initial time **BEGIN_METEO_DATA_(HOURS_AFTER_00)**.
- **Z_CUTS_(M)**: Terrain following heights (in m) at which maps are produced.

TIME.UTC

```

YEAR = 2007
MONTH = 03
DAY = 01
BEGIN_METEO_DATA_(HOURS_AFTER_00) = 0
END_METEO_DATA_(HOURS_AFTER_00) = 3
TIME_STEP_METEO_DATA_(MIN) = 60

```

DATABASE_GRID

```

UTM_ZONE = 33
UTM_HEMISPHERE = N
X_ORIGIN_(UTM_M) = 475000
Y_ORIGIN_(UTM_M) = 4100000
CELL_SIZE_(KM) = 1.0
NX = 101
NY = 101
Z_LAYER_(M) = 0 10 40 100 250 500 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000

```

CALMET

```

MESOSCALE_MODEL = AMITA
MESOSCALE_TIME_INCREMENT_(HOURS) = 3
MESOSCALE_RANGE_OF_LATITUDES = 35.0 40.0
MESOSCALE_RANGE_OF_LONGITUDES = 12.0 15.0

```

POSTPROCESS_DBS

```

OUTPUT_FILES_IN_GRD_FORMAT = YES
OUTPUT_FILES_IN_PS_FORMAT = YES
POSTPROCESS_TIME_INTERVAL_(HOURS) = 3
Z_CUTS_(M) = 1000. 5000.

```

Table 1: Example of a meteorological database input file. In this example (see section 8) a database of 100x100x10 km is created. It contains 101x101x16=163216 points with hourly meteorological data from 01 MAR 2007 at 0000UTC to 01 MAR at 0300UTC.

2.4 The database script files

In order to automate the creation/update of databases the APOLLO procedure contains a series of script files that control the flow of a database construction. These files are, obviously, operating system dependent and are located in the folder *Scripts/Db*s. The default folder structure and file names defined in section 7 are assumed.

- Script APOLLO-Build-Db-s-Db-sName. Controls the creation of a database named Db-sName. There must exist a different script file for each database to create/update. This script updates first the date in the database input file (through a call to a secondary script named APOLLO-Db-s-touchdate) and then controls the construction of the meteorological database (this task is done by the script APOLLO-Db-s-engine).
- Script APOLLO-Db-s-touchdate. Changes the date in the date in the database input file.
- Script APOLLO-Db-s-engine. Does the calls to the programs GEOINP, MESOINP, CALMETINP, CALMET, BUILDDBS, and POSTPDBS. The user can control which programs have to run.

3 Run generation

3.1 Overview

A run is the simulation of a given scenario. APOLLO allows to include several fallout models within the same run. Since models may run using different number of particle classes (that is, with a different discretization of granulometry) and/or using different spatial discretizations, it is necessary to supply different source and granulometry files to each model. A run is defined in the run input file (see section 3.5) and executed through a script file (see section 3.6). For each model present in a run the flow includes:

1. Creation of the granulometry file using the program SETGRANUM. Alternatively the same file can be supplied by the user without running the SETGRANUM utility.
2. Creation of a source file using the program SETSRC. Alternatively the same file can be supplied by the user without running the SETSRC utility.
3. Model run.
4. Optionally, model postprocess using the program MODELPOSTP.

3.2 Generation of a granulometry file. The program SETGRANUM

DESCRIPTION: The granulometric distribution for a model is stored in a granulometry file (see section 6.3 for details). The program SETGRANUM is an utility that reads the **GRANULOMETRY** block of a run input file (see section 3.5) and generates a granulometry file assuming that the mass fraction of particles follows a Gaussian distribution in Φ and that the density and the sphericity of particles vary linearly with Φ . Note that, in general, each model present in a run has to have its own granulometry file because the number of discrete particle classes may vary from model to model. Note that other distributions different from Gaussian and having arbitrary density-size and sphericity-size relationships can be also considered. In this case, however, the granulometry files can not be generated by SETGRANUM but must be supplied directly by the user.

PROGRAM CALL: (normally included in a script file): Path of the executable + 4 arguments

`"SetGranum.exe FileLog FileRunInp FileGrn ModelName"`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileRunInp:** Path (including name) of the run input file (see Section 3.5) that contains the **GRANULOMETRY** block.
- **FileGrn:** Name (including path) of the granulometry file. This is the output from SETGRANUM that is used later by models.
- **ModelName:** Name of the model (must coincide with the corresponding model block in the run input file). This is used to read the number of granulometric classes used by each model.

3.3 Generation of a source file. The program SETSRC

DESCRIPTION: The distribution of sources is defined in a source file (see section 6.4 for details). The program SETSRC is an utility that reads the **SOURCE** block from the control input file (see section 3.5) and generates a source file. The source term is constant for a given time interval but there is no limit on the number and duration of the time intervals. It allows, in practise, to discretize any kind of time-dependency (time-dependent mass flow rate, column height, etc.). The program admits three possibilities: point source (mass is released in a single source point), Suzuki distribution (Suzuki , 1983; Pfeiffer et al., 2005), and buoyant plume model (Bursik , 2001). The last option is more elaborated and involves the solution of the 1D radial-averaged plume governing equations that describe the convective region of an eruptive column. These equations are intimately coupled with the wind field which, for small to medium size plumes, may induce a substantial plume bent-over and subsequent variations of plume height and mass release location. For this reason, when this option switched on, the program reads the values of the wind field from a meteorological file, computes the averaged wind direction and solves the plume governing equations for each time interval and particle class accounting for wind. Note that it introduces a time dependence in the source term even when all the eruptive parameters (mass flow rate, class fraction, etc.) are kept constant in time.

PROGRAM CALL: (normally included in a script file): Path of the executable + 7 arguments

“SetSrc.exe FileLog FileRunInp FileSrc FileGrn FileDBs ModelName UseMesh”

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileRunInp:** Path (including name) of the run input file (see Section 3.5) that contains the **SOURCE** block.
- **FileSrc:** Name (including path) of the source file. This is the output from SETSRC that is used later by models.
- **FileGrn:** Name (including path) of the granulometry file.
- **FileDBs:** Path (including name) of the DBs file. It is a binary file created by the BUILDDBS program. Only used if the **SOURCE_TYPE** record in the run input file is **PLUME**.
- **ModelName:** Name of the model (must coincide with the corresponding model block in the run input file). This is used to read the number of granulometric classes used by each model.
- **UseMesh:** Flag to indicate if the results are projected onto the model mesh. Possibilities are YES or NO.

3.4 Fallout models

3.4.1 HAZMAP model

DESCRIPTION: HAZMAP is a FORTRAN90 code for the solution of the equation of diffusion, transport and sedimentation of small particles, in order to model the dispersion of ash generated by a convective column. Under the approximations of a constant horizontally uniform wind field, and negligible vertical advection and diffusion, this equation reduces to:

$$\frac{\partial C_j}{\partial t} + u_x \frac{\partial C_j}{\partial x} + u_y \frac{\partial C_j}{\partial y} - \frac{\partial (v_{sj} C_j)}{\partial z} = K \left(\frac{\partial^2 C_j}{\partial x^2} + \frac{\partial^2 C_j}{\partial y^2} \right) + S_j \quad (1)$$

where C_j is the concentration of the particle-class j having a settling velocity v_{sj} , $\mathbf{u} = (u_x, u_y)$ is the wind velocity, K is the (constant) horizontal turbulent diffusion coefficient, and S_j is the source term. The generic particle class j is defined by triplet of values characterizing each particle (d_p, ρ_p, F_p) , that are, respectively, diameter, density, and a shape factor. For d_p we use the equivalent diameter d , which is the diameter of a sphere of equivalent volume. For the shape factor F_p we choose the sphericity ψ , which is the ratio of the surface area of a sphere with diameter d to the surface area of the particle. In our approximation, each triplet (d, ρ_p, ψ) is sufficient to define the settling velocity value v_{sj} . Since eq.(1) is linear in mass, an instantaneous release of the total mass from the eruption column can be assumed if wind and diffusion parameters do not change significantly with time and only the final deposit is needed. This quasi-steady approach is assumed to hold during each simulation time interval. Considering these approximations, the above equation has a semi-analytical solution as described in (Macedonio et al., 2005). The computational domain is split into thin horizontal layers that fall to the ground together with the particles originally contained in a given initial vertical interval $[z_i, z_{i+1}]$ at time $t = 0$. An analytical solution is then found for each layer. Since the whole treatment is done separately for each class of particles and no vertical diffusion and wind advection takes place, all particles falling from the same initial height remain at all times at the same altitude. While the centre of each cloud is translated by wind, the cloud spreads horizontally due to diffusion and settles by gravity until it reaches the ground where it forms the deposit. The model outputs therefore accumulations on the ground for each granulometric class. For further details see (Macedonio et al., 2005) and (Pfeiffer et al., 2005). There are several semi-empirical parameterizations for the particle settling velocity v_s if one assumes that particles settle down at their terminal velocity:

$$v_s = \sqrt{\frac{4g(\rho_p - \rho_a)d}{3C_d\rho_a}} \quad (2)$$

where ρ_a and ρ_p denote air and particle density, respectively, d is the particle equivalent diameter, and C_d is the drag coefficient. C_d depends on the Reynolds number, $Re = dv_s/\nu_a$ ($\nu_a = \mu_a/\rho_a$ is the kinematic viscosity of air, μ_a the dynamic viscosity). In HAZMAP several options are possible for estimating settling velocity, such as:

1. **ARASTOPOUR** model (Arastoopour et al., 1982):

$$C_d = \begin{cases} \frac{24}{Re}(1 + 15Re^{0.687}) & Re \leq 10^3 \\ 0.44 & Re > 10^3 \end{cases} \quad (3)$$

valid for spherical particles only.

2. **GANSER** model (Ganser, 1993):

$$C_d = \frac{24}{ReK_1} \left\{ 1 + 0.1118 [Re(K_1K_2)]^{0.6567} \right\} + \frac{0.4305K_2}{1 + \frac{3305}{ReK_1K_2}} \quad (4)$$

where $K_1 = 3/(1 + 2\psi^{-0.5})$, $K_2 = 10^{1.84148(-\text{Log}\psi)^{0.5743}}$ are two shape factors, and ψ is the particle sphericity ($\psi = 1$ for spheres).

3. **WILSON** model (Wilson and Wang, 1979) using the interpolation suggested by Pfeiffer et al. (2005):

$$C_d = \begin{cases} \frac{24}{Re}\varphi^{-0.828} + 2\sqrt{1.07 - \varphi} & Re \leq 10^2 \\ 1 - \frac{1 - C_d|_{Re=10^2}}{900}(10^3 - Re) & 10^2 \leq Re \leq 10^3 \\ 1 & Re \geq 10^3 \end{cases} \quad (5)$$

where $\varphi = (b + c)/2a$ is the particle aspect ratio ($a \geq b \geq c$ denote the particle semi-axes).

4. DELLINO model (Dellino et al., 2005):

$$v_s = 1.2605 \frac{\nu_a}{d} (Ar \xi^{1.6})^{0.5206} \quad (6)$$

where $Ar = d^3(\rho_p - \rho_a)\rho_a/\mu_a^2$ is the Archimedes number, and ξ is a particle shape factor (sphericity to circularity ratio).

PROGRAM CALL (normally included in a script file): Path of the executable + 7 arguments

“Hazmap.exe FileLog FileRunInp FileSrc FileGrn FileDbbs FileLst FileRes”

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileRunInp:** Path (including name) of the run input file (see Section 3.5) that contains the HAZMAP block.
- **FileSrc:** Name (including path) of the source file.
- **FileGrn:** Name (including path) of the granulometry file.
- **FileDbbs:** Path (including name) of the Dbbs file. It is a binary file created by the BUILDDBS program.
- **FileLst:** Path (including name) of the Lst file. It is an output ASCII file with information about the HAZMAP run.
- **FileRes:** Path (including name) of the Res file. This is binary output file with the results of HAZMAP. For file format see section 6.5.

3.4.2 FALL3D model

DESCRIPTION: FALL3D is a 3D time-dependent Eulerian model which circumvents most of the simplifications behind the simpler fallout models. The model solves the advection-diffusion-sedimentation equation using a finite differences explicit scheme using a regular mesh (Costa et al., 2005). It uses the gradient transport theory to evaluate the atmospheric turbulent diffusion within and above the ABL and experimental fits for the particle settling velocities, in addition to the values of the dataset (full 3D prognostic wind field, source term and topography). The model can be therefore used to forecast either ash concentration in the atmosphere or ash loading on the ground. APOLLO contains both serial and parallel versions of FALL3D. The structure of the code combined with the fact that the interaction among particles is a second order effect facilitates the parallelization enormously. Two kinds of parallelization are considered, one for particle classes and one for space (vertical layers). Firstly, the processors available are distributed among groups or teams. Each team works only on a certain particle class or on a set of particle classes (the number of processors must be, in consequence, a multiple or a divisor of the number of classes). If each particle class has more than one processor assigned (*i.e.* if the number of processors is a multiple of the number of classes) a second parallelization on the domain is possible. In this case, the tasks within a team are subdivided in vertical layers. Note that it implies a data exchange among processors of the same team but the teams remain isolated among them.

The non-conservative form of continuity equation written in a generalised coordinate system (X, Y, Z) is:

$$\begin{aligned} \frac{\partial C}{\partial t} + U \frac{\partial C}{\partial X} + V \frac{\partial C}{\partial Y} + (W - V_{sj}) \frac{\partial C}{\partial Z} = -C \nabla \cdot \mathbf{U} + C \frac{\partial V_{sj}}{\partial Z} \\ \frac{\partial}{\partial X} \left(\rho_* K_X \frac{\partial C / \rho_*}{\partial X} \right) + \frac{\partial}{\partial Y} \left(\rho_* K_Y \frac{\partial C / \rho_*}{\partial Y} \right) + \frac{\partial}{\partial Z} \left(\rho_* K_Z \frac{\partial C / \rho_*}{\partial Z} \right) + S_* \end{aligned} \quad (7)$$

where C is the scaled average concentration, (U, V, W) are the scaled wind speeds, K_X, K_Y and K_Z are the diagonal scaled diffusion coefficients, ρ_* the scaled atmospheric density and S_* is the source term in the generalized coordinate system. Considering as a frame of reference a simple terrain-following coordinate system where the horizontal coordinates remain unchanged with respect to the Cartesian $(x = X, y = Y, z \rightarrow Z)$, the scaling factors are those reported in Table 2. Equation (7) is solved for each particle class independently, i.e. assuming no interaction between particles belonging to different classes during the transport process. The generic particle class j is defined by triplet of values characterizing each particle (d_p, ρ_p, F_p) , that are, respectively, diameter, density, and a shape factor. For d_p we use the equivalent diameter d , which is the diameter of a sphere of equivalent volume. For the shape factor F_p we choose the sphericity ψ , which is the ratio of the surface area of a sphere with diameter d to the surface area of the particle. In our approximation, each triplet (d, ρ_p, ψ) is sufficient to define the settling velocity value V_{sj} . Settling velocity models available in FALL3D are ARASTOPOUR (see eq. 3), GANSER (see eq. 4), WILSON (see eq. 5), and DELLINO (see eq. 6). For a detailed description on the numeric algorithm see Costa et al. (2005).

In order to solve equation (7) it is necessary to evaluate the vertical and horizontal diffusion coefficients. Inside the atmospheric surface layer, the Monin-Obukhov similarity theory estimates the vertical turbulent diffusivity K_z in terms of the friction velocity u_* , and the Monin-Obukhov length L :

$$K_z = \frac{\kappa z u_*}{\phi_h} \quad (8)$$

where κ is the von Karman constant ($\kappa = 0.4$), z is the distance from the ground, and ϕ_h is the atmospheric stability function (e.g. Jacobson, 1999). Above the surface layer, the original form of the Monin-Obukhov similarity theory is no longer valid. In order to extend this theory to the entire boundary layer ($z/h < 1$) an evaluation of the Atmospheric Boundary Layer (ABL) height h is required. For this purpose, FALL3D uses a simple parameterisation valid on the entire ABL (Ulke, 2000):

$$K_z = \begin{cases} \kappa u_* z \left(1 - \frac{z}{h}\right) \left(1 + 9.2 \frac{h}{L} \frac{z}{h}\right)^{-1} & h/L \geq 0 \quad \text{stable} \\ \kappa u_* z \left(1 - \frac{z}{h}\right) \left(1 - 13 \frac{h}{L} \frac{z}{h}\right)^{1/2} & h/L \leq 0 \quad \text{unstable} \end{cases} \quad (9)$$

Note that in the neutral case ($L \rightarrow \infty$) both expressions coincide. Finally, in the free atmosphere above the ABL ($z/h > 1$), K_z is considered a function of the local vertical wind gradient, a characteristic length scale l_c , and a stability function F_c depending on the Richardson number Ri :

$$K_z = l_c^2 \left| \frac{\partial V}{\partial z} \right| F_c(Ri) \quad (10)$$

For l_c and F_c the model adopts the relationship used by the CAM3 model (Collins et al., 2004) of the National Center for Atmospheric Research (NCAR):

$$l_c = \left(\frac{1}{\kappa z} + \frac{1}{\lambda_c} \right)^{-1} \quad (11)$$

$$F_c(Ri) = \begin{cases} \frac{1}{1 + 10Ri(1 + 8Ri)} & \text{stable } (Ri > 0) \\ \frac{1}{\sqrt{1 - 18Ri}} & \text{unstable } (Ri < 0) \end{cases} \quad (12)$$

where λ_c is the so-called asymptotic length scale ($\lambda_c \approx 30\text{m}$) while the Richardson number is calculated as $Ri = \frac{g}{\theta_v} \frac{\partial\theta_v/\partial z}{|\partial V/\partial z|^2}$ (with θ being virtual potential temperature).

On the other hand, for the horizontal eddy diffusivity $K_H = K_x = K_y$ FALL3D assumes a large eddy parameterisation as that used by RAMS model (for $\Delta z/\Delta \ll 1$ Pielke et al., 1992):

$$K_H = R \max \left(K_{mh}; (C_{SH}\Delta)^2 \sqrt{\left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}\right)^2 + 2 \left[\left(\frac{\partial v_x}{\partial x}\right)^2 + \left(\frac{\partial v_y}{\partial y}\right)^2 \right]} \right) \quad (13)$$

$$K_{mh} = 0.075 K_A \Delta^{4/3}$$

where $\Delta = \sqrt{\Delta_x \Delta_y}$, C_{SH} is a dimensionless constant ranging from 0.135 to 0.32, K_A is a user defined parameter close to one, and $R \simeq 3$.

PROGRAM CALL (serial version): Path of the executable + 7 arguments

`“Fall3d_ser.exe FileLog FileRunInp FileSrc FileGrn FileDbS FileLst FileRes”`

PROGRAM CALL (parallel version): Path of the executable + 8 arguments

`“Fall3d_par.exe FileLog FileRunInp FileSrc FileGrn FileDbS FileLst FileRes
Ncpu”`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileRunInp:** Path (including name) of the run input file (see Section 3.5) that contains the FALL3D block.
- **FileSrc:** Name (including path) of the source file.
- **FileGrn:** Name (including path) of the granulometry file.
- **FileDbS:** Path (including name) of the DbS file. It is a binary file created by the BUILDDBS program.
- **FileLst:** Path (including name) of the Lst file. It is an output ASCII file with information about the FALL3D run.
- **FileRes:** Path (including name) of the Res file. This is binary output file with the results of FALL3D. For file format see section 6.5.
- **Ncpu:** Number of CPU groups.

3.4.3 TEPHRA model

DESCRIPTION: TEPHRA (Connor et al., 2001) is an Eulerian model based on an analytical solution of eq. (1). The particle fall time depends on particle properties (density, diameter) and

Parameter	Scaling
Coordinates	$X = x \quad Y = y \quad Z = z - h(x, y)$
Velocities	$U = u_x \quad V = u_y \quad W = u_z J^{-1} \quad V_{sj} = v_{sj} J^{-1}$
Diffusion Coefficients	$K_X = K_x \quad K_Y = K_y \quad K_Z = K_z J^{-2}$
Concentration	$C = cJ$
Density	$\rho_* = \rho J$
Source Term	$S_* = SJ$

Table 2: Summary of the scaling factors for the terrain-following domain coordinate system ($x = X$, $y = Y$, $z \rightarrow Z$). J indicates the Jacobian of the coordinate system transformation.

atmospheric density. Settling velocity is determined assuming spherical particles and considering different regimes depending on the Reynolds number and atmospheric density as:

$$V_s = \begin{cases} \frac{\rho_a g d^2}{18\mu} & Re \leq 6 \quad (laminar) \\ d \left(\frac{4g^2 \rho_p}{255\mu \rho_a} \right)^{1/3} & 6 \leq Re \leq 500 \quad (intermediate) \\ \sqrt{\frac{3.1gd\rho_p}{\rho_a}} & Re \geq 500 \quad (turbulent) \end{cases} \quad (14)$$

where ρ_a and ρ_p stand, respectively, for the air and particle densities, d is the particle diameter, μ is air viscosity, and Re is the Reynolds number. Note that, in the turbulent regime, this coincides with equation (3) (spherical particles). On the other hand, diffusion of particles in the atmosphere is estimated using a bivariate Gaussian probability density function to approximate turbulence with the variance σ given by:

$$\sigma = \begin{cases} 4Kt + 0.0128z^2 & t > t_o \quad (coarse \ particles) \\ \frac{8}{5}c \left(t + (0.2z^2)^{2/5} \right)^{5/2} & t \leq t_o \quad (fine \ particles) \end{cases} \quad (15)$$

where t is the total particle fall time, t_o is a threshold time, z is the particle release height, and c is a constant.

PROGRAM CALL (normally included in a script file): Path of the executable + 7 arguments

`“Tephra.exe FileLog FileRunInp FileSrc FileGrn FileDbS FileLst FileRes”`

- **FileLog**: Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileRunInp**: Path (including name) of the run input file (see Section 3.5) that contains the TEPHRABlock.
- **FileSrc**: Name (including path) of the source file.
- **FileGrn**: Name (including path) of the granulometry file.
- **FileDbS**: Path (including name) of the DbS file. It is a binary file created by the BUILDDBS program.
- **FileLst**: Path (including name) of the Lst file. It is an output ASCII file with information about the TEPHRA run.
- **FileRes**: Path (including name) of the Res file. This is binary output file with the results of TEPHRA. For file format see section 6.5.

3.5 The run input file

A run input file (see Table 3) is an ASCII file composed by a series of blocks and labels. Labels are case sensitive and can be placed in any order within a block. Comments and extra lines can be inserted anywhere. This file controls the input parameters needed by the programs SETGRANUM, SETSRC, MODELPOSTP as well as by the different models (HAZMAP, FALL3D and TEPHRA). Each model has its own block (labelled like the model) where the model inputs are specified. A new block named `MODELNAME` should be added to this file whenever a new model is added to the APOLLO runs. There must exist an input file for each run.

Block TIME.UTC (read by SETSRC and the models). This block contains labels that define the time range of a run (in UTC time). The run time interval must lay within the time interval bounds of the meteorological database to which a run is linked.

- **YEAR:** Simulation year.
- **MONTH:** Simulation month (1-12).
- **DAY:** Simulation day (1-31).
- **RUN_START_(HOURS_AFTER_00):** Run start hour (after 0000UTC of current day).
- **ERUPTION_END_(HOURS_AFTER_00) :** Eruption end hour (after 0000UTC of current day). If the SETSRC program is used to generate the source term, this is the time instant at which source is switched off.
- **RUN_END_(HOURS_AFTER_00):** Run end hour (after 0000UTC). Note that, in general, a run can continue even when the source term is switched off (*i.e.* when the eruption has ceased) in order to give time for the remaining airborne particles to fall. This can be important in time-dependent models such as FALL3D. In contrast, for steady or quasi-steady models (HAZMAP or TEPHRA) this is irrelevant and **RUN_END_(HOURS_AFTER_00)** can coincide with **ERUPTION_END_(HOURS_AFTER_00)**.

Block SOURCE (read by SETSRC). This block contains labels that define the source characteristics.

- **X_VENT_(UTM_M):** x -coordinate of the vent (UTM coordinates in m).
- **Y_VENT_(UTM_M):** y -coordinate of the vent (UTM coordinates in m).
- **MASS_FLOW_RATE_(KGS):** Values of the mass flow rate (in kg/s). One value for each time interval. The duration of each time interval is constant and given by **RUN_START_(HOURS_AFTER_00)** minus **ERUPTION_END_(HOURS_AFTER_00)** divided by the number of time intervals (automatically computed by the program from the number of values).
- **SOURCE_TYPE:** Type of source distribution. Possibilities are **POINT**, **SUZUKI** or **PLUME**.
- **HEIGHT_ABOVE_VENT_(M):** Heights of the plume (in m above the vent). One value for each time interval.
- **A:** Parameter A in the Suzuki distribution. One value for each time interval. Used only if **SOURCE_TYPE = SUZUKI**.
- **L:** Parameter L in the Suzuki distribution. One value for each time interval. Used only if **SOURCE_TYPE = SUZUKI**.

- **EXIT_VELOCITY_(MS)**: Magma exit velocity (in m/s) at the vent. One value for each time interval. Used only if **SOURCE_TYPE** = **PLUME**.
- **EXIT_TEMPERATURE_(K)**: Magma exit temperature (in $^{\circ}K$) at the vent. One value for each time interval. Used only if **SOURCE_TYPE** = **PLUME**.
- **EXIT_VOLATILE_FRACTION_(IN%)**: Magma volatile mass fraction at the vent. One value for each time interval. Used only if **SOURCE_TYPE** = **PLUME**.

Block GRANULOMETRY (read by SETGRANUM). This block contains labels that define the granulometric characteristics.

- **NUMBER_OF_CLASSES**: Number of particle classes.
- **FI_MEAN**: Mean value of Φ (Gaussian distribution).
- **FI_DISP**: Value of σ in the Gaussian distribution.
- **FI_RANGE**: Minimum and maximum values of Φ (Φ_{min} and Φ_{max} respectively).
- **DENSITY_RANGE**: Values of density (in kg/m^3) associated to Φ_{min} and Φ_{max} particles. Linear interpolation is assumed.
- **SPHERICITY_RANGE**: Values of sphericity associated to Φ_{min} and Φ_{max} particles. Linear interpolation is assumed.

Block FALL3D (read by the FALL3D model). This block contains labels that define the FALL3Dinput data.

- **ZLAYER_(M)**: Heights (in m) of the z -layers in terrain following coordinates, *i.e.* above the vent. It is not necessary to specify the number of vertical layers since it is automatically calculated by the program. Alternatively, for regular z -layering, the user can also specify the initial value (z_o), the final value (z_f), and the increment (Δz) using the format:
ZLAYER_(M) FROM z_o TO z_f INCREMENT Δz
- **NUMBER_OF_CLASSES**: Numbner of particle classes.
- **TERMINAL_VELOCITY_MODEL**: Type of terminal settling velocity model. Possibilities are **ARASTOPOUR** (Arastoopour et al., 1982), **GANSER** (Ganser , 1993), **WILSON** (Wilson and Wang , 1979) and **DELLINO** (Dellino et al., 2005).
- **VERTICAL_TURBULENCE_MODEL**: Type of model for vertical diffusion. Possibilities are **CONSTANT** or **SIMILARITY**. See section 3.4.2 and Costa et al. (2005) for details.
- **VERTICAL_DIFFUSION_COEFFICIENT_(M2/S)**: Value of the diffusion coefficient (in m^2/s). Only used if **VERTICAL_TURBULENCE_MODEL** = **CONSTANT**
- **HORIZONTAL_TURBULENCE_MODEL**: Type of model for horizontal diffusion. Possibilities are **CONSTANT** or **RAMS**. See section 3.4.2 and Costa et al. (2005) for details.
- **HORIZONTAL_DIFFUSION_COEFFICIENT_(M2/S)**: Value of the diffusion coefficient (in m^2/s). Only used if **HORIZONTAL_TURBULENCE_MODEL** = **CONSTANT**.
- **POSTPROCESS_TIME_INTERVAL_(HOURS)**: Time interval to output results (in h). Results are also output at the end of the run.

Block HAZMAP (read by the HAZMAP model). This block contains labels that define the HAZMAP input data.

- **ZLAYER(M)**: Heights (in m) of the z -layers in terrain following coordinates, *i.e.* above the vent. It is not necessary to specify the number of vertical layers since it is automatically calculated by the program. Alternatively, for regular z -layering, the user can also specify the initial value (z_o), the final value (z_f), and the increment (Δz) using the format:
ZLAYER(M) FROM z_o TO z_f INCREMENT Δz
- **NUMBER_OF_CLASSES**: Number of particle classes.
- **TERMINAL_VELOCITY_MODEL**: Type of terminal settling velocity model. Possibilities are ARASTOPOUR (Arastoopour et al., 1982), GANSER (Ganser, 1993), WILSON (Wilson and Wang, 1979) and DELLINO (Dellino et al., 2005).
- **HORIZONTAL_DIFFUSION_COEFFICIENT_(M2/S)**: Value of the diffusion coefficient K (in m^2/s).
- **POSTPROCESS_TIME_INTERVAL_(HOURS)**: Time interval to output results (in h).

Block TEPHRA (read by the TEPHRA model). This block contains labels that define the TEPHRA input data.

- **ZLAYER(M)**: Heights (in m) of the z -layers in terrain following coordinates, *i.e.* above the vent. It is not necessary to specify the number of vertical layers since it is automatically calculated by the program. Alternatively, for regular z -layering, the user can also specify the initial value (z_o), the final value (z_f), and the increment (Δz) using the format:
ZLAYER(M) FROM z_o TO z_f INCREMENT Δz
- **NUMBER_OF_CLASSES**: Number of particle classes.
- **DIFFUSION_COEFFICIENT_(M2/S)**: Value of the diffusion coefficient K (in m^2/s).
- **FALL_TIME_THRESHOLD**: Value of fall time threshold t_o (see eq. (15)).
- **EDDY_CONSTANT**: Value of constant c (see eq. (15)).

Block POSTPROCESS_MODELS (read by MODELPOSTP). This block contains labels used to define the postprocess of models and production of maps. It is only read if the optional program MODELPOSTP runs.

- **OUTPUT_FILES_IN_GRD_FORMAT**: Possibilities are YES or NO. If YES, MODELPOSTP plots files in GRD format. Files in GRD format (see section 6.6 for details) can be read directly by several plotting programs like the commercial software GRAPHER. Alternatively, the user may also generate its own plots using functions from several free packages (*e.g.* gnuplot in FORTRAN).
- **OUTPUT_FILES_IN_PS_FORMAT**: Possibilities are YES or NO. If YES, MODELPOSTP plots files in PS format.
- **MAP_TOTAL_LOAD**: Possibilities are YES or NO. If YES MODELPOSTP plots the total ground load.
- **UNITS**: Units of MAP_TOTAL_LOAD. It must be KG/M2.
- **CONTOUR_LEVELS**: Values of the contour levels for MAP_TOTAL_LOAD. Only used when OUTPUT_FILES_IN_PS_FORMAT is YES.

- **MAP_CLASS_LOAD**: Possibilities are YES or NO. If YES MODELPOSTP plots the class ground load.
- **UNITS**: Units of MAP_CLASS_LOAD. It must be KG/M2.
- **CONTOUR_LEVELS**: Values of the contour levels for MAP_CLASS_LOAD. Only used when OUTPUT_FILES_IN_PS_FORMAT is YES.
- **MAP_DEPOSIT_THICKNESS**: Possibilities are YES or NO. If YES MODELPOSTP plots total deposit thickness.
- **UNITS**: Units of MAP_DEPOSIT_THICKNESS. Possibilities are MM (for *mm*), CM (for *cm*), and M (for *m*).
- **COMPACTATION_FACTOR**: Deposit compactation factor.
- **CONTOUR_LEVELS**: Values of the contour levels for MAP_DEPOSIT_THICKNESS. Only used when OUTPUT_FILES_IN_PS_FORMAT is YES.
- **MAP_TOTAL_CONCENTRATION**: Possibilities are YES or NO. If YES MODELPOSTP plots the total concentration at certain *z*-levels. Only for 3D models (FALL3D).
- **UNITS**: Units of MAP_TOTAL_CONCENTRATION. It must be KG/M3.
- **Z_CUTS_(M)**: *z*-coordinates of the layers at which concentration is output.
- **CONTOUR_LEVELS**: Values of the contour levels for MAP_TOTAL_CONCENTRATION. Only used when OUTPUT_FILES_IN_PS_FORMAT is YES.
- **MAP_Z_CUMMULATIVE_CONCENTRATION**: Possibilities are YES or NO. If YES MODELPOSTP plots the *z* cummulative concentration (vertical integration). Only for 3D models (FALL3D).
- **UNITS**: Units of MAP_Z_CUMMULATIVE_CONCENTRATION. It must be KG/M2.
- **CONTOUR_LEVELS**: Values of the contour levels for MAP_Z_CUMMULATIVE_CONCENTRATION. Only used when OUTPUT_FILES_IN_PS_FORMAT is YES.
- **MAP_Z_MAXIMUM_CONCENTRATION**: Possibilities are YES or NO. If YES MODELPOSTP plots the maximum value of concentration along the vertical for each point. This variable can be useful for flight safety concentration tresholds. Only for 3D models (FALL3D).
- **UNITS**: Units of MAP_Z_MAXIMUM_CONCENTRATION. It must be KG/M3.
- **CONTOUR_LEVELS**: Values of the contour levels for MAP_Z_MAXIMUM_CONCENTRATION. Only used when OUTPUT_FILES_IN_PS_FORMAT is YES.

Table 3: Example run input file (see section 8).

```

TIME.UTC
YEAR = 2007
MONTH = 03
DAY = 01
ERUPTION_START_(HOURS_AFTER_00) = 0
ERUPTION_END_(HOURS_AFTER_00) = 0.1
RUN_END_(HOURS_AFTER_00) = 2.

SOURCE
X_VENT_(UTM_M) = 500080.
Y_VENT_(UTM_M) = 4177690.
MASS_FLOW_RATE_(KGS) = 3d6 (One value for each source time interval)
SOURCE_TYPE = PLUME
POINT_SOURCE (Variables below used if SOURCE_TYPE = POINT)
HEIGHT_ABOVE_VENT_(M) = 8000.
SUZUKI_SOURCE (Variables below used if SOURCE_TYPE = SUZUKI)
HEIGHT_ABOVE_VENT_(M) = 8000. (One value for each source time interval)
A = 4. (One value for each source time interval)
L = 5. (One value for each source time interval)
PLUME_SOURCE (Variables below used if SOURCE_TYPE = PLUME)
EXIT_VELOCITY_(MS) = 100.
EXIT_TEMPERATURE_(K) = 1073.
EXIT_VOLATILE_FRACTION_(IN%) = 0.

GRANULOMETRY
FI_MEAN = 2.5
FI_DISP = 1.5
FI_RANGE = 0. 5.
DENSITY_RANGE = 1200 2300
SPHERICITY_RANGE = 0.9 0.9

FALL3D
Z_LAYER_(M) FROM 0. TO 10000. INCREMENT 500.
NUMBER_OF_CLASSES = 6
TERMINAL_VELOCITY_MODEL = GANSER
VERTICAL_TURBULENCE_MODEL = SIMILARITY
VERTICAL_DIFFUSION_COEFFICIENT_(M2/S) = 100. (if VERTICAL_TURBULENCE_MODEL = CONSTANT)
HORIZONTAL_TURBULENCE_MODEL = RAMS
HORIZONTAL_DIFFUSION_COEFFICIENT_(M2/S) = 2500. (if HORIZONTAL_TURBULENCE_MODEL = CONSTANT)
POSTPROCESS_TIME_INTERVAL_(HOURS) = 1.

```

HAZMAP

Z_LAYER_(M) FROM 0. TO 10000. INCREMENT 250.
 NUMBER_OF_CLASSES = 12
 TERMINAL_VELOCITY_MODEL = Ganser
 HORIZONTAL_DIFFUSION_COEFFICIENT_(M2/S) = 2500.
 POSTPROCESS_TIME_INTERVAL_(HOURS) = 1.

TEPHRA

Z_LAYER_(M) FROM 0. TO 10000. INCREMENT 250.
 NUMBER_OF_CLASSES = 12
 DIFFUSION_COEFFICIENT_(M2/S) = 2500.
 FALL_TIME_THRESHOLD = 2500.0
 EDDY_CONSTANT = 0.03

POSTPROCESS_MODELS

OUTPUT_FILES_IN_GRD_FORMAT = YES
 OUTPUT_FILES_IN_PS_FORMAT = YES
 MAP_TOTAL_LOAD = YES
 UNITS = KG/M2
 CONTOUR_LEVELS = 0.1 0.25 0.5 1. 5. 10. 50. (Only used if OUTPUT_FILES_IN_PS_FORMAT=YES)
 MAP_CLASS_LOAD = NO
 UNITS = KG/M2
 CONTOUR_LEVELS = 0.1 0.25 0.5 1. 5. 10. 50. (Only used if OUTPUT_FILES_IN_PS_FORMAT=YES)
 MAP_DEPOSIT_THICKNESS = YES
 UNITS = MM
 COMPACTION_FACTOR = 0.7
 CONTOUR_LEVELS = 0.1 1. 5. 10. 50. 100. 500. (Only used if OUTPUT_FILES_IN_PS_FORMAT=YES)
 MAP_TOTAL_CONCENTRATION = YES
 UNITS = KG/M3
 Z_CUTS_(M) = 1000. 2000.
 CONTOUR_LEVELS = 1d-5 1d-4 (Only used if OUTPUT_FILES_IN_PS_FORMAT=YES)
 MAP_Z_CUMMULATIVE_CONCENTRATION = YES
 UNITS = KG/M2
 CONTOUR_LEVELS = 0.01 0.1 1. 10. (Only used if OUTPUT_FILES_IN_PS_FORMAT=YES)
 MAP_Z_MAXIMUM_CONCENTRATION = YES
 UNITS = KG/M3
 CONTOUR_LEVELS = 1d-4 1d-3 (Only used if OUTPUT_FILES_IN_PS_FORMAT=YES)

3.6 The run script files

In order to automate the execution of models the APOLLO procedure contains a series of script files that control the run flow. These files are, obviously, operating system dependent. The default folder structure and file names defined in section 7 are assumed.

- Scripts APOLLO-Run-ProblemName. These scripts (located in the folder *Scripts/Runs*) control the run of a problem named ProblemName. There must exist a different script for each problem. The scripts update first the date in the run input file (through a call to a secondary script named APOLLO-Run-touchdate) and then control the run of each model (this task is done by other scripts, one for model, named APOLLO-ModelName-engine). The user can control which models have to run.
- Script APOLLO-Run-touchdate. This script (located in the folder *Scripts/Runs*) changes the date in the run input file.
- Scripts APOLLO-ModelName-engine. These scripts (located in the folder *Scripts/Models*) run a certain model (*e.g.* APOLLO-Hazmap-engine runs HAZMAP) calling first the program SETGRANUM to generate the model granulometry, then the program SETSRC to generate the source term, then the model and, finally, the program MODELPOSTP to postprocess results.

4 Postprocess of models

4.1 Overview

Fallout models output either 2D results at ground surface (normally deposit load and/or deposit thickness) or 3D results (concentration on air). In many cases it is interesting to evaluate also other derived variables that may have interest from the point of view of hazard assessment or crisis management. For example, the air-borne ash burden serves to compare simulations with satellital images, or the maximum value of concentration along the vertical can give insights on flight safety if the volcanic cloud moves towards the vicinity of an airport or intersects an aerial corridor. From a practical point of view it is better to split the postprocess computations from the model run (that is, to calculate these quantities after the model run).

4.2 The program MODELPOSTP

DESCRIPTION: The program MODELPOSTP (alias for Model Postprocess) is an optional utility that reads a model output binary file (see section 6.5), calculates some relevant quantities at selected z -planes and time instants and produces elementary maps in GRD (see section 6.6) and PS formats. The parameters needed by MODELPOSTP are defined in the block POSTPROCESS_MODELS, located at the end of the run input file.

PROGRAM CALL (normally included in a script file): Path of the executable + 4(5) arguments

`"ModelPostp.exe FileLog FileRunInp FileRes BASERES (FileSym)"`

- **FileLog:** Path (including name) of the log file. It is an ASCII file that contains information about the program execution.
- **FileRunInp:** Path (including name) of the run input file (see Section 3.5) that contains the POSTPROCESS_MODELS block.
- **FileRes:** Path (including name) of the Res file. This is binary output file with the results of models. For file format see section 6.5.
- **BASERES:** Path where the MODELPOSTP output files are dump.
- **FileSym:** Path (including name) of the symbols file. This file is optional (MODELPOSTP can be called using either 4 or 5 arguments) and is used to add symbols and legends to the PS files. For file format see section 6.7.

5 The library LIBAPOLLO

LIBAPOLLO is a library written in FORTRAN 77. It contains a set of user-callable routines that act as an interface between programs and files used by the APOLLO procedure. When invoked within a program, these routines allow to read and extract information from different files. The programs included in the APOLLO procedure make use of LIBAPOLLO routines for file read operations. The use of this library is strongly recommended (although not mandatory) if the user wishes to add a new model to the procedure. The reason is twofold. First because it simplifies enormously the codes. Second, and more important, because any future change in the format of a file will imply modifications only in the library, but not in the programs/models, which will remain unchanged. The LIBAPOLLO contains several families of routines devoted to different purposes.

NOTE: The routines of the library can be called either from FORTRAN or C using the same syntax. To call routines from C simply include the header CtoF.h (`#include "CtoF.h"`) located in the folder `/Programs/LibApollo/CtoF`.

5.1 Routines to read an input control file

An input control file (either for a meteorological database or for a specific run) is an ASCII file composed by blocks of lines (labels). Each block starts with a header that informs about the general contents of the lines below. In turn, each line of a block starts with a header word that can be followed by a number of words and/or parameters (a parameter is a number, either integer or real). Line headers within a block can not be repeated, but two lines can be identical if they belong to different blocks. Comments can be placed anywhere. The routines that read an input file search first for a specific block header and then for a specific line header within the block. The rest of the file contents is simply ignored.

- **subroutine APOLLO_get_input_npar**

PURPOSE: Gets the number of parameters (numbers) included in a line.

SINTAX: call APOLLO_get_input_npar (fname, block, line, npar, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
block	INPUT	char	any	Block header
line	INPUT	char	any	Line header
npar	OUTPUT	int	4	Number of parameters found in the line
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_input_nword**

PURPOSE: Gets the number of words included in a line.

SINTAX: call APOLLO_get_input_nword (fname, block, line, nword, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
block	INPUT	char	any	Block header
line	INPUT	char	any	Line header
nword	OUTPUT	int	4	Number of words found in the line (header not included)
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_input_int4**

PURPOSE: Reads nval integers of length 4 from a line.

SINTAX: call APOLLO_get_input_int4 (fname, block, line, value, nval, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
block	INPUT	char	any	Block header
line	INPUT	char	any	Line header
nval	INPUT	int	4	Number of values to read
value	OUTPUT	int(nval)	4	Values of the nval integers to read
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_input_rea4**

PURPOSE: Gets nval reals of length 4 from a line.

SINTAX: call APOLLO_get_input_rea4 (fname, block, line, value, nval, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
block	INPUT	char	any	Block header
line	INPUT	char	any	Line header
nval	INPUT	int	4	Number of values to read
value	OUTPUT	rea(nval)	4	Values of the nval reals to read
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_input_rea8**

PURPOSE: Gets nval reals of length 8 from a line.

SINTAX: call APOLLO_get_input_rea8 (fname, block, line, value, nval, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
block	INPUT	char	any	Block header
line	INPUT	char	any	Line header
nval	INPUT	int	4	Number of values to read
value	OUTPUT	rea(nval)	8	Values of the nval reals to read
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_input_cha**

PURPOSE: Gets nval characters from a line.

SINTAX: call APOLLO_get_input_cha (fname, block, line, value, nval, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
block	INPUT	char	any	Block header
line	INPUT	char	any	Line header
nval	INPUT	int	4	Number of values to read
value	OUTPUT	char(nval)	any	Values of the nval strings to read
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

5.2 Routines to read a database

A database is composed by properties and meteorological variables. Properties, which can be integer or real numbers, define characteristics of the database like date, location, number of

points, etc. Meteorological variables are the contents of the database and are stored in a direct access file. There exists a file record for each variable at a particular time instant and for each vertical layer. The length of a record is therefore $nx \times ny$ (8 bytes).

- **subroutine APOLLO_get_dbs_property_int4**

PURPOSE: Gets the value of an integer type property from a database.

SINTAX: call APOLLO_get_dbs_property_int4 (fname,word,value,istat,message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
word	INPUT	char	any	Code of the property to read. Possibilities are "DATE": Date of the database (YYYYMMDD) "BEGIN": Initial time of data (<i>h</i> after 00UTC) "END": Final time of data (<i>h</i> after 00UTC) "NX": Number of points in the <i>x</i> -direction "NY": Number of points in the <i>y</i> -direction "NZ": Number of vertical layers
value	OUTPUT	int	4	Value of the property defined in word
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_dbs_property_rea8**

PURPOSE: Gets the value of an real type property from a database.

SINTAX: call APOLLO_get_dbs_property_rea8 (fname,word,value,istat,message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
word	INPUT	char	any	Code of the property to read. Possibilities are "X-ORIGIN": <i>x</i> -origin coordinate (UTM in <i>m</i>) "Y-ORIGIN": <i>y</i> -origin coordinate (UTM in <i>m</i>) "X-SPACE": <i>x</i> -grid spacing (in <i>km</i>) "Y-SPACE": <i>y</i> -grid spacing (in <i>km</i>) "Z-LAYERS": <i>z</i> -layers coordinates (in <i>m</i>)
value	OUTPUT	real	8	Value of the property defined in word. Returns an scalar except if word="Z-LAYERS". In this case returns nz values
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_dbs_value_point**

PURPOSE: Gets the value of a variable on a point and at a given time instant. The point coordinates must lay within the bounds of the database. If the point does not coincide with a node of the database grid, results are interpolated bilinearly. A terrain following coordinate system is assumed.

SINTAX: call APOLLO_get_dbs_value_point (fname, timesec, word, x, y, z, value, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in <i>s</i> after 00UTC) at which data is extracted
word	INPUT	char	any	Code of the property to read. Possibilities are TOPOGRAPHY VELOCITY-X VELOCITY-Y VELOCITY-Z TEMPERATURE CONVECTIVE-VELOCITY-SCALE MIXING-HEIGHT MONIN-OBUKHOV-LENGTH
x	INPUT	real	8	Point <i>x</i> -coordinate (UTM in <i>m</i>)
Y	INPUT	real	8	Point <i>y</i> -coordinate (UTM in <i>m</i>)
z	INPUT	real	8	Point <i>z</i> -coordinate (in <i>m</i> , terrain following) NOTE: $z=0$ for TOPOGRAPHY, CONVECTIVE-VELOCITY-SCALE, MIXING-HEIGHT and MONIN-OBUKHOV- LENGTH
value	OUTPUT	real	8	Value of the property defined in word
endsec	OUTPUT	int	4	Time (in <i>s</i> after 00UTC) until which the value of the variable remains unchanged
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_dbs_value_plane**

PURPOSE: Gets the values of a variable on a plane (vertical layer) at a given time instant. The plane is assumed to have $nx \times ny$ points, at the same horizontal location that those of the database. If the plane *z*-coordinate does not coincide with a layer of the database, results are interpolated linearly. The plane can be above the upper limit of the database. A terrain following coordinate system is assumed.

SINTAX: call APOLLO_get_dbs_value_plane (fname, timesec, word, z, value, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in <i>s</i> after 00UTC) at which data is extracted
word	INPUT	char	any	Code of the property to read. Possibilities are TOPOGRAPHY VELOCITY-X VELOCITY-Y VELOCITY-Z TEMPERATURE CONVECTIVE-VELOCITY-SCALE MIXING-HEIGHT MONIN-OBUKHOV-LENGTH
z	INPUT	real	8	Plane <i>z</i> -coordinate (in <i>m</i> , terrain following) NOTE: $z=0$ for TOPOGRAPHY, CONVECTIVE-VELOCITY-SCALE, MIXING-HEIGHT and MONIN-OBUKHOV-LENGTH
value	OUTPUT	real(<i>nx*ny</i>)	8	Values at the $nx \times ny$ nodes of the plane
endsec	OUTPUT	int	4	Time (in <i>s</i> after 00UTC) until which the value of the variable remains unchanged
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if <i>istat</i> \neq 0)

5.3 Routines to read a source file

A source file contains information about the distribution of the mass flow rate (source term) for each granulometric class. The number and position of sources may vary with time. See section 6.4 for details on file format.

- **subroutine APOLLO_get_source_nsrc**

PURPOSE: Gets the number of sources.

SINTAX: call APOLLO_get_source_nsrc (fname, timesec, nsrc, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in <i>s</i> after 00UTC) at which data is extracted
nsrc	OUTPUT	int	4	Number of sources
endsec	OUTPUT	int	4	Time (in <i>s</i> after 00UTC) until which the value of the source term remains unchanged
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if <i>istat</i> \neq 0)

- **subroutine APOLLO_get_source_nclass**

PURPOSE: Gets the number of classes.

SINTAX: call APOLLO_get_source_nclass (fname, timesec, nclass, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in <i>s</i> after 00UTC) at which data is extracted
nclass	OUTPUT	int	4	Number of classes
endsec	OUTPUT	int	4	Time (in <i>s</i> after 00UTC) until which the value of the source term remains unchanged
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_source_coordinates**

PURPOSE: Gets the coordinates of the sources (nsrc values) on a terrain following coordinate system.

SINTAX: call APOLLO_get_source_coordinates (fname, timesec, x, y, z, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in <i>s</i> after 00UTC) at which data is extracted
x	OUTPUT	real(nsrc)	8	<i>x</i> -coordinates of the source points. It returns nsrc values.
y	OUTPUT	real(nsrc)	8	<i>y</i> -coordinates of the source points. It returns nsrc values.
z	OUTPUT	real(nsrc)	8	<i>z</i> -coordinates of the source points. It returns nsrc values.
endsec	OUTPUT	int	4	Time (in <i>s</i> after 00UTC) until which the value of the source term remains unchanged
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_source_value**

PURPOSE: Gets the values of the source points (nsrc x nclass values).

SINTAX: call APOLLO_get_source_value (fname, timesec, src, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in <i>s</i> after 00UTC) at which data is extracted
src	OUTPUT	real(nsrc*nclass)	8	Source values.
endsec	OUTPUT	int	4	Time (in <i>s</i> after 00UTC) until which the value of the source term remains unchanged
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

5.4 Routines to read a granulometry file

A granulometry file contains information about particle classes: granulometry, fraction, density, and sphericity. It is assumed constant during the whole run. See section 6.3 for details on file format.

- **subroutine APOLLO_get_granulometry_nclass**

PURPOSE: Gets the number of granulometric classes.

SINTAX: call APOLLO_get_granulometry_nclass (fname, nc, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
nc	OUTPUT	int	4	Number of granulometric classes
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_get_granulometry_value**

PURPOSE: Gets a granulometric property (nc values).

SINTAX: call APOLLO_get_granulometry_value (fname, word, val, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
word	INPUT	char	any	Code of the property to read. Possibilities are DIAMETER. Returns diameters (in mm) DENSITY. Returns densities (in kg/m^3). SPHERICITY. Returns particle sphericities. FRACTION. Returns mass fractions.
val	OUTPUT	real(nc)	8	nc values of the property defined by word
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

5.5 Routines to read a wind profile file

A wind profile file contains wind velocity and temperature at different heights and time intervals. See section 6.2 for details on file format.

- **subroutine APOLLO_get_wind_profile_point**

PURPOSE: Gets wind properties at a height z .

SINTAX: call APOLLO_get_wind_profile_point (fname, timesec, z, ux, uy, T, endsec, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
timesec	INPUT	int	4	Time (in s after 00UTC) at which data is read
z	INPUT	real	8	Height at which data is read
ux	OUTPUT	real	8	Wind x -velocity (m/s)
uy	OUTPUT	real	8	Wind y -velocity (m/s)
T	OUTPUT	real	8	Air temperature (in $^{\circ}K$)
endsec	OUTPUT	int	4	Time (in s after 00UTC) until which data is valid
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

5.6 Routines to output model results

- **subroutine APOLLO_out_model_result2d**

PURPOSE: Writes model results (deposit load or thickness) at surface.

SINTAX: call APOLLO_out_model_result2d(fname,header0,header1,icode,nx,ny,itime,var,istat,message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
header0	INPUT	char	any	Free header
header1	INPUT	char	any	Free header
icode	INPUT	int	4	Variable code. Possibilities are: 0 for TOTAL DEPOSIT LOAD 1 for DEPOSIT THICKNESS -i for CLASS i DEPOSIT LOAD
nx	INPUT	int	4	Number of points in the x -direction
ny	INPUT	int	4	Number of points in the y -direction
itime	INPUT	int	4	Time (in s) after 00UTC
var	INPUT	real(nx*ny)	8	Variable to output
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_out_model_result3d**

PURPOSE: Writes 3D model results (concentration on air).

SINTAX: call APOLLO_out_model_result3d(fname, header0, header1, icode, nx, ny, nz, itime, var, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
header0	INPUT	char	any	Free header
header1	INPUT	char	any	Free header
icode	INPUT	int	4	Variable code. Possibilities are: 1 for AIR CONCENTRATION
nx	INPUT	int	4	Number of points in the x -direction
ny	INPUT	int	4	Number of points in the y -direction
nz	INPUT	int	4	Number of points in the z -direction
itime	INPUT	int	4	Time (in s) after 00UTC
var	INPUT	real(nx*ny*nz)	8	Variable to output
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

- **subroutine APOLLO_out_model_structuredgrid**

PURPOSE: Outputs model geometry for postprocess. Grid is assumed to be structured with uniform spacings along x and y directions. Spacing along the vertical direction z can vary.

SINTAX: call APOLLO_out_model_structuredgrid(fname, nx, ny, nz, xo, yo, dx, dy, dz, top, istat, message)

Variable		kind	length	Description
fname	INPUT	char	any	Path (including name) of the file
nx	INPUT	int	4	Number of points in the x -direction
ny	INPUT	int	4	Number of points in the y -direction
nz	INPUT	int	4	Number of points in the z -direction
xo	INPUT	real	8	x -coordinate (UTM in m) of the origin (bottom left corner)
yo	INPUT	real	8	y -coordinate (UTM in m) of the origin (bottom left corner)
dx	INPUT	real	8	Grid spacing (in m) along the x -direction
dy	INPUT	real	8	Grid spacing (in m) along the y -direction
dz	INPUT	real(nz)	8	Grid spacings (in m) along the z -direction
top	INPUT	real(nx*ny)	8	Topography
istat	OUTPUT	int	4	Execution status. 0 means no error
message	OUTPUT	char	100	Output message (only if istat \neq 0)

6 File formats

6.1 The terrain file format

DESCRIPTION: ASCII file containing terrain information at discrete points of a regular 2D grid. Points are ordered in lines of constant y , from W to E. In turn, lines are ordered from N to S. It normally covers a large area (*e.g.* 1000x1000 km). A database may always lay within the bounds of the terrain file. It must be created by the user or downloaded from the APOLLO project website.

FORMAT:

Line 1	Free header
Line 2	nx ny xo yo dx dy
Line 3	Free header
Lines 4 to 4+ (nx x ny)	x y z ldu zo alb bow shf ahf leaf

where:

- **nx:** Number of cells in the x -direction.
- **ny:** Number of cells in the y -direction.
- **xo:** x -coordinate of the grid bottom left corner (UTM coordinates in m).
- **xf:** x -coordinate of the grid top right corner (UTM coordinates in m).
- **yo:** y -coordinate of the grid bottom left corner (UTM coordinates in m).
- **yf:** y -coordinate of the grid top right corner (UTM coordinates in m).
- **z:** topography. In m .
- **ldu:** Land use according to the USGS convention.
- **alb:** Albedo at point (x, y) .
- **shf:** Soil heat flux at point (x, y) .
- **ahf:** Antropogenic heat flux at point (x, y) .
- **leaf:** Leaf index at point (x, y) .

6.2 The wind profile file format

DESCRIPTION: ASCII file containing the definition of the vertical wind profile and air temperature at different time intervals. This file can be read using LIBAPOLLO routines (see section 5.5).

FORMAT:

itime1 itime2
nz
z(1) ux(1) uy(1) T(1)
...
z(nz) ux(nz) uy(nz) T(nz)
...

where:

- **itime1**: Starting time (in sec after 00UTC) of the meteo data time slice.
- **itime2**: End time (in sec after 00UTC) of the meteo data time slice.
- **nz**: Number of vertical layers.
- **z**: Vertical coordinate of the layer (in *m*, terrain following).
- **ux**: wind *x*-velocity (in *m/s*).
- **uy**: wind *y*-velocity (in *m/s*).
- **T**: Air temperature (in $^{\circ}C$).

6.3 The granulometry file format

DESCRIPTION: The granulometry file is an ASCII file containing the definition of the particle classes (a class is characterized by particle size, density and sphericity). This file can be created by the utility program SETGRANUM and read using LIBAPOLLO routines (see section 5.4).

FORMAT:

```
nc
diam(1) rho(1) shpe(1) fc(1)
...
diam(nc) rho(nc) sphe(1) fc(nc)
```

where:

- **nc**: Number of granulometric classes.
- **diam**: Class diameter (in *mm*).
- **rho**: Class density (in kg/m^3).
- **sphe**: Class sphericity.
- **fc**: Class mass fraction (0-1). It must verify that $\sum fc = 1$.

6.4 The source file format

DESCRIPTION: The source file is an ASCII file containing the definition of the source term. The source is defined at time intervals during which source values are kept constant. The number, position and values (*e.g.* Mass Flow Rate) of the source points can, however, vary from one time slice to another. There is no restriction on the number and duration of the time intervals. It allows, in practise, to discretize any type of source term. This file can be created by the utility program SETSRC and read using LIBAPOLLO routines (see section 5.3).

FORMAT:

```

itime1 itime2
nsrc nc
MFR
x y z src(1,1) ... src(1,nc)
...
x y z src(nsrc,1) ... src(nsrc,nc)
...
```

where:

- **itime1**: Starting time (in sec after 00UTC) of the time interval.
- **itime2**: End time (in sec after 00UTC) of the time interval.
- **nsrc**: Number of source points (can vary from one interval to another).
- **nc**: Number of granulometric classes.
- **MFR**: Mass flow rate (in kg/s).
- **x**: x -coordinate of the source $isrc$ (UTM coordinates in m).
- **y**: y -coordinate of the source $isrc$ (UTM coordinates in m).
- **z**: z -coordinate of the source $isrc$ (terrain following coordinates in m , *i.e.* above the vent).
- **src**: Mass flow rate (in kg/s) of each granulometric class for this point source. It must be verified that $\sum \sum src(isrc, ic) = MFR$.

6.5 The model output file format

DESCRIPTION: This is a binary file with the results from models. It is assumed that results are output at the nodes of a regular 2d or 3d grid. Models must output results in this format if they are to be processed by the MODELPOSTP utility. These can be done using LIBAPOLLO routines (see section 5.6).

FORMAT: The file contains first a model grid block followed by number of results blocks (one block for each output quantity and time instant in the case of transient models). The model grid block contains three records with the following quantities:

```

record 1  nx, ny, nz, xo, yo, dx, dy
record 2  zlayer(nz)
record 3  topg(nx,ny)
```

whereas each block of results contains 4 records with:

```

record 1  idime, icode, itime, lenh1, lenh2
record 2  header1
record 3  header2
record 4  Results
```

where:

- **nx**: Number of cells in the x -direction.
- **ny**: Number of cells in the y -direction.

- **xo**: x -coordinate of the grid bottom left corner (UTM coordinates in m).
- **yo**: y -coordinate of the grid bottom left corner (UTM coordinates in m).
- **dx**: Grid spacing (in m) along the x -direction.
- **dy**: Grid spacing (in m) along the y -direction.
- **zlayer**: Coordinates of the grid vertical layers (terrain following in m).
- **topg**: Topography.
- **idime**: Spatial dimensions of results. It can be 2 or 3 for results on a plane or in the space respectively.
- **icode**: Code for results. Possibilities if $idime=2$ are:
 - $icode < 0$. Deposit load for granulometric class ABS($icode$).
 - $icode = 0$. Total deposit load.
 - $icode = 1$. Deposit thickness.

whereas possibilities if $idime=3$ are:

- $icode < 0$. Concentration for granulometric class ABS($icode$).
- $icode = 1$. Total concentration on air.
- **itime**: Time of results. Given in seconds after 00UTC for the current day.
- **lenh1**: Length of the header1.
- **lenh2**: Length of the header2.
- **header1**: Free header for comments. In the models HAZMAP, FALL3D and TEPHRA contains the description of the results.
- **header2**: Free header for comments. In the models HAZMAP, FALL3D and TEPHRA contains the date and time in format YYYY:MM:DD:HH:SSSS.
- **results**: These are $nx \times ny$ real*8 values if $idime=2$ and $nx \times ny \times nz$ real*8 values if $idime=3$.

6.6 The GRD file format

DESCRIPTION: ASCII grid files (.GRD) contain results in a 2D structured grid.

FORMAT: GRD files contain five header lines that provide information about the size and limits of the grid, followed by a list of z -values (scalar variable). The fields within ASCII grid files must be space delimited. The listing of z -values follows the header information in the file. The z -values are stored in row-major order starting with the minimum y -coordinate. The first z -value in the grid file corresponds to the lower left corner of the map. This can also be thought of as the southwest corner of the map, or, more specifically, the grid node of minimum x and minimum y . The second z -value is the next adjacent grid node in the same row (the same y -coordinate but the next higher x -coordinate). When the maximum x -value is reached in the row, the list of z -values continues with the next higher row, until all the rows of z -values have been included. The general format of an ASCII grid file is:

```

DSAA
nx ny
xo xf
yo yf
zmin zmax
z(1,1) ... z(1,nx)
...
z(ny,1) ... z(ny,nx)

```

where:

- **nx**: Number of cells in the x -direction.
- **ny**: Number of cells in the y -direction.
- **xo**: x -coordinate of the grid bottom left corner (UTM coordinates in m).
- **xf**: x -coordinate of the grid top right corner (UTM coordinates in m).
- **yo**: y -coordinate of the grid bottom left corner (UTM coordinates in m).
- **yf**: y -coordinate of the grid top right corner (UTM coordinates in m).
- **zmin**: Minimum value of z in the domain.
- **zmax**: Maximum value of z in the domain.
- **z**: Value of z at each grid point.

6.7 The symbols file format

DESCRIPTION: ASCII file that contains the symbols and legends added to the PS format postprocess files. This file is used (optionally) by the program MODELPOSTP and specifies the coordinates of relevant geographic features (*e.g.* cities, airports, etc.). If these fall within the bounds of the computational domain the program MODELPOSTP adds the legend and the associated symbol to the PS files.

FORMAT:

```

x y legend code flag
...
x y legend code flag
...

```

where:

- **x**: x -coordinate.
- **y**: y -coordinate.
- **legend**: Word that defines the geographic feature (*e.g.* Catania).
- **code**: An integer number that defines the code of the symbol associated to the feature (see Figure 2).
- **flag**: Integer flag to switch on/off this particular feature. If **flag** is 1 MODELPOSTP adds the feature to the PS file.

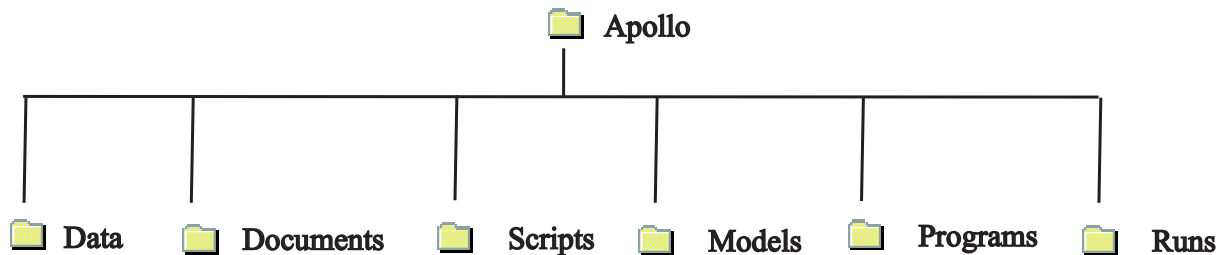
EXAMPLE: To add a squared symbol with the legend “Catania” add the following line to the file: 507000. 4152000. Catania 157 1

Characters and octal codes for Font ZapfDingbats																
40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57	
60	61	62	63	64	65	66	67	70	71	72	73	74	75	76	77	
100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117	
120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137	
140	141	142	143	144	145	146	147	150	151	152	153	154	155	156	157	
160	161	162	163	164	165	166	167	170	171	172	173	174	175	176	177	
240	241	242	243	244	245	246	247	250	251	252	253	254	255	256	257	
260	261	262	263	264	265	266	267	270	271	272	273	274	275	276	277	
300	301	302	303	304	305	306	307	310	311	312	313	314	315	316	317	
320	321	322	323	324	325	326	327	330	331	332	333	334	335	336	337	
340	341	342	343	344	345	346	347	350	351	352	353	354	355	356	357	
360	361	362	363	364	365	366	367	370	371	372	373	374	375	376		

Figure 2: List of symbols.

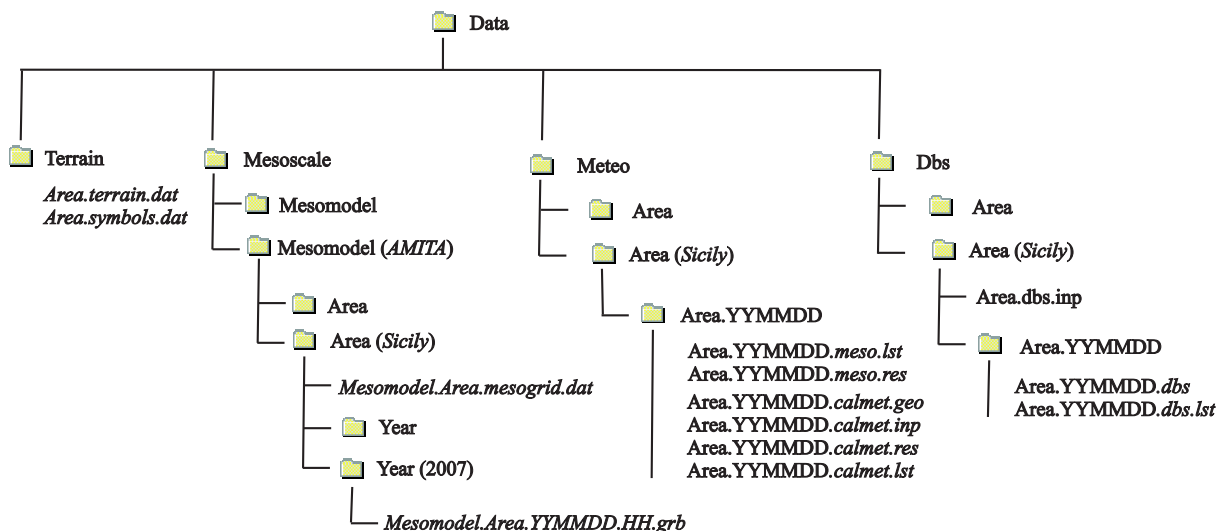
7 The default APOLLO tree

Once installed, the *apollo* directory contains 6 folders: *Data*, *Documents*, *Master*, *Models*, *Programs*, and *Runs*. The location and names of files created during the execution of programs and models is set in the scripts (scripts call the programs and pass the full paths of files to be created as a program call argument). It is recommended to keep the default APOLLOtree structure. However, if a user wishes to change file names and locations it is sufficient to modify the scripts accordingly.

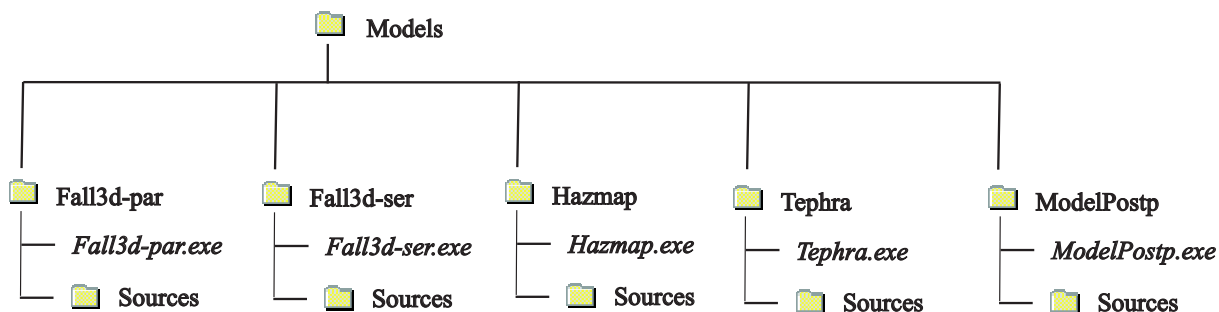


1. Folder *Data*. Contains the terrain and meteorological data.

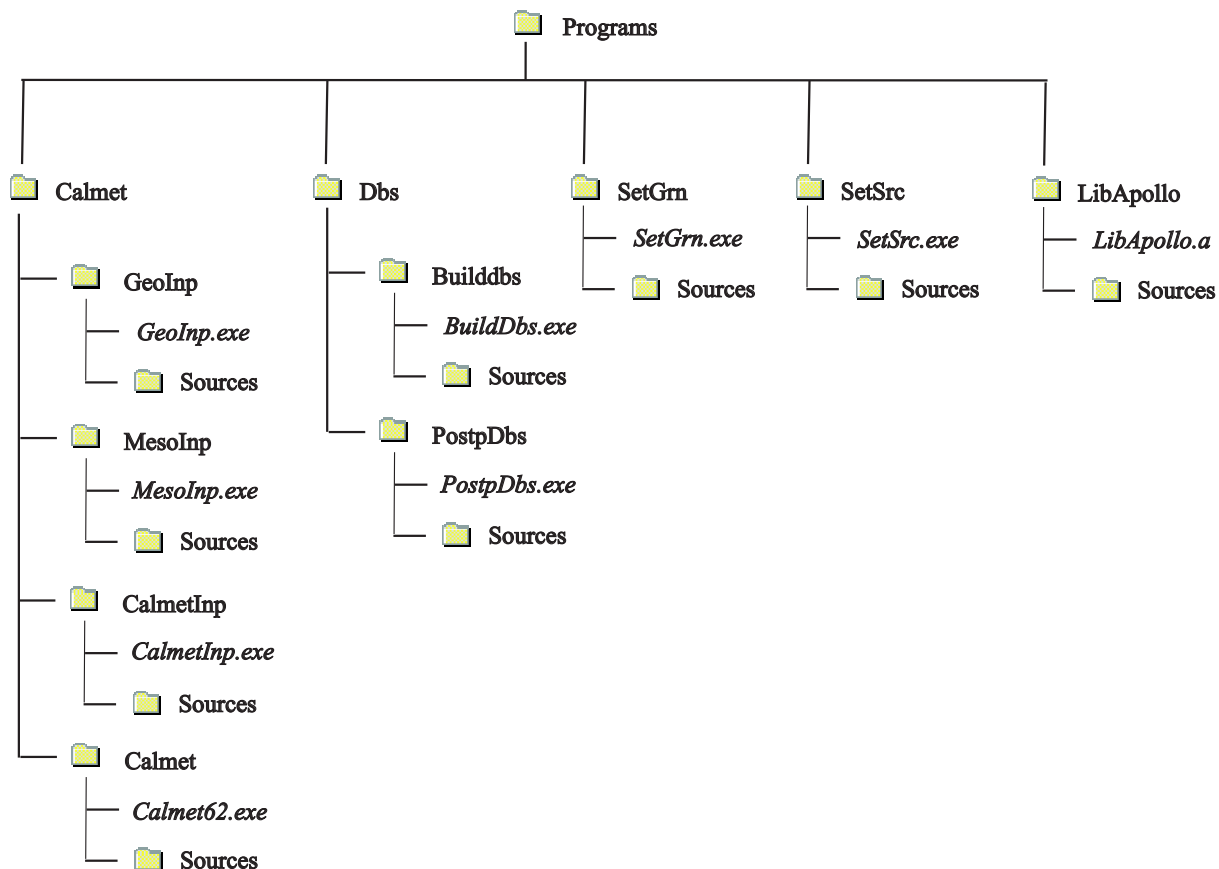
- (a) Folder *Terrain*. Contains the terrain files *Area.terrain.dat* (see section 6.1) and the symbols files (see section 6.7).
- (b) Folder *Mesoscale*. Contains the mesoscale meteorological predictions. Each mesoscale model has its own folder which, in turn, can have different *Area* folders. The latter contain the mesoscale meteorological grid for the Mesomodel.*Area.mesogrid.dat*.
- (c) Folder *Meteo*. Contains the results of the CALMET runs, including CALMET input and output files. It is not necessary to keep these files since the meteorological data are, in practice, stored in the database.
- (d) Folder *Dbs*. Contains the meteorological databases. Each area has its own folder where the database input file *Area.dbs.inp* for this particular area resides. By default, periodic (daily) updates are not deleted. Thus, a folder *Area.YYMMDD* is created every day (YYMMDD stands for YearMonthDay; e.g. 070120 for January 20th 2007) to store the database files.



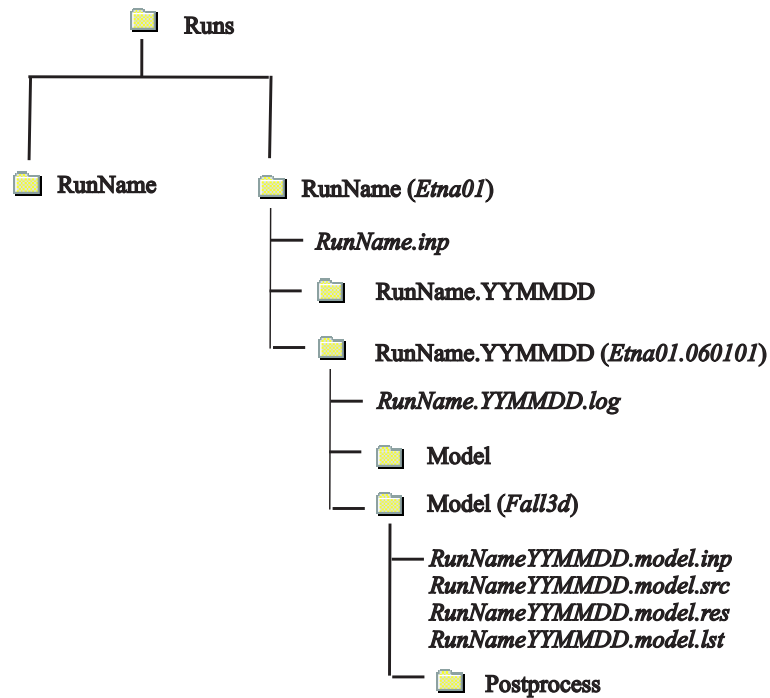
2. Folder *Documents*. Contains APOLLO documentation, including this manual.
3. Folder *Scripts*. Contains the scripts. These files are, obviously, OS dependent.
4. Folder *Models*. Contains the source codes and the executables of the fallout models and the program MODELPOSTP. Models added by users should lay in this folder.



5. Folder *Programs*. Contains the source codes and the executables of the APOLLO programs and the library LIBAPOLLO.



6. Folder *Runs*. Contains the runs. Each run has a folder with the run input file and the results for the different models. By default, periodic (daily) runs are not deleted. Thus, a folder *RunName.YYMMDD* is created every day (YYMMDD stands for YearMonthDay; e.g. 070120 for January 20th 2007) to store the run results.



8 Application example

The APOLLO package includes an application example to check that the installation and compilation of the procedure has been done successfully. The example considers an eruption occurring at the first of March 2007 with characteristics similar to those of the paroxystic phase of the 22 July 1998 Mt. Etna eruption. The files needed to run the example are:

- Sicily.terrain.dat (located in the folder *Data/Terrain/*). This is a terrain file (see section 6.1) for the area of Sicily. Needed by the program GEOINP.
- AMITA.Sicily.070301.hh.grb (located in the folder *Data/Mesoscale/AMITA/Sicily/2007/*). These are GRIB-format files coming from the AMITA (Aeronautica Militare ITALiana) mesoscale model, and furnish meteo data from 01 MAR 2007 at 0000UTC to 01 MAR 2007 at 0600UTC. Needed by the program MESOINP.
- AMITA.Sicily.mesogrid.dat (located in the folder *Data/Mesoscale/AMITA/Sicily/*). Contains information on the AMITA mesoscale model grid. Needed by the program MESOINP.
- Sicily.inp (located in the folder *Data/DbS/Sicily/*). This is the database input file (see Table 1) that defines the characteristics of a database named “Sicily”.
- Etna.inp (located in the folder *Runs/Etna/*). This is the problem run input file (see Table 3) that defines the characteristics of a run named “Etna”.

To run the example it is necessary to proceed as follows:

1. Launch the script APOLLO-Build-DbS-Sicily (located in the folder */Scripts/DbS/*) to create the database Sicily. This script calls the programs GEOINP, MESOINP, CALMETINP, CALMET, BUILDDBS, and POSTPDBS.
2. Launch the script APOLLO-Run-Etna (located in the folder */Scripts/Runs/*) to run the fallout models.

Figure 3 shows the deposit load predicted by the different models. Figure 4 shows the z -cumulative concentration predicted by FALL3D at time instants 0100UTC and 0200UTC.

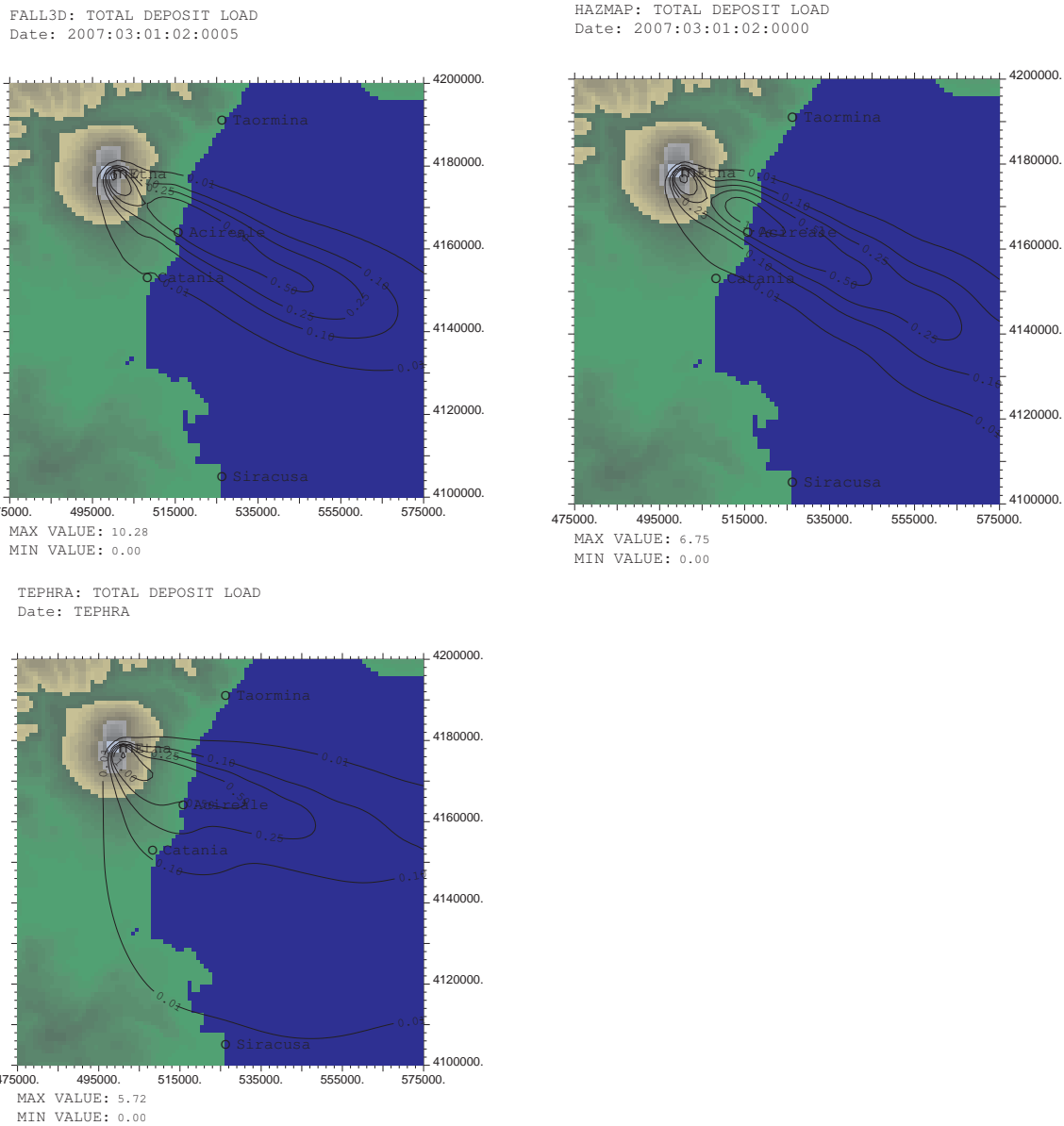


Figure 3: Deposit load (in kg/m^2) at 0200UTC. Results for FALL3D, HAZMAP, and TEPHRA.

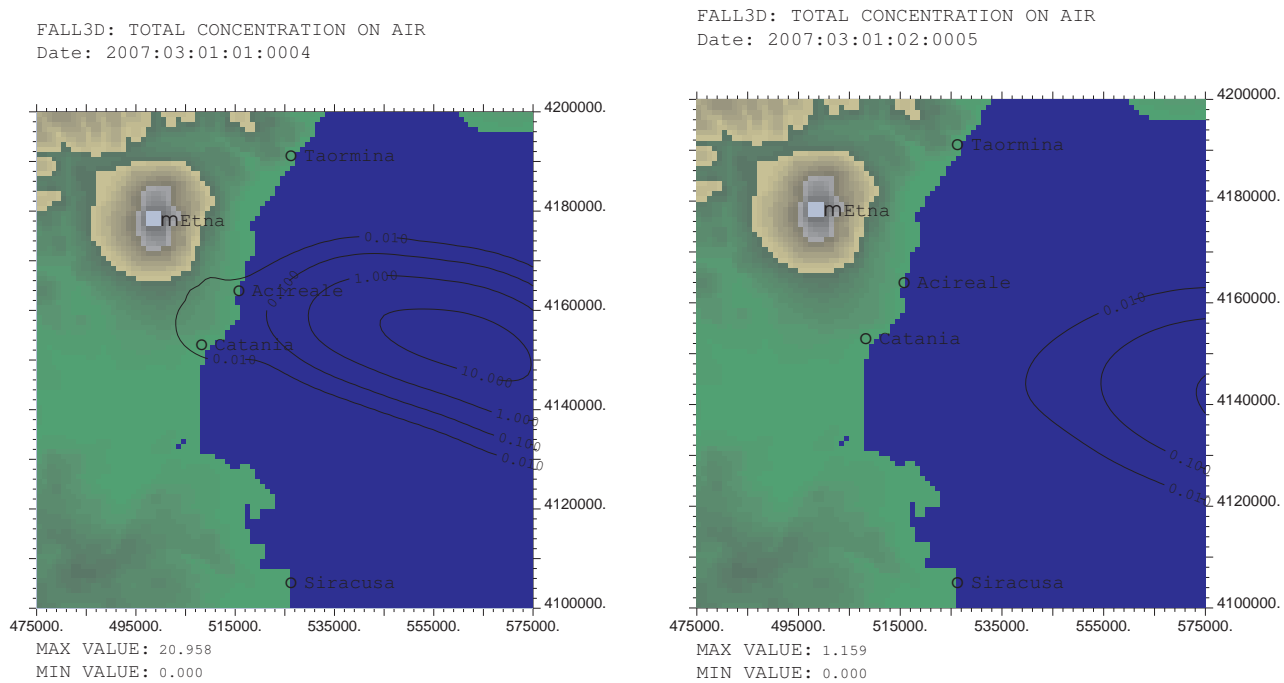


Figure 4: FALL3D model. z -cumulative concentration at 0100UTC and 0200UTC.

9 References

- Arastoopour, H., Wang, C., Weil, S., 1982. Particle-particle interaction force in a dilute gassolid system. *Chemical Engineering Science* 37, 1379–1386.
- Azad, A., Kitada, T., 1998. Characteristic of the air pollution in the city of Dhaka, Bangladesh in winter. *Atmos. Environ.* 32, 1991–2005.
- Bursik, M., 2001. Effect of wind on the rise height of volcanic plumes. *Geophys. Res. Lett.* 18, 3621–3624.
- Casadevall, T. J., 1993. Volcanic Ash and Airports - Discussion and Recommendations from the Workshop on Impacts of Volcanic Ash on Airport Facilities. U.S. Geological Survey Open-File Report, 93-518, 52 pp.
- Chester, D.L., Degg, M., Duncan, A.M., Guest, J.E., 2001. The increasing exposure of cities to the effects of volcanic eruptions: a global survey. *Environ. Haz.*, 2, 89–103.
- Collins, W., Rasch, P., Boville, B., Hack, J., McCaa, J., Williamson, D., Kiehl, J., Briegleb, B., 2004. Description of the NCAR Community Atmosphere Model (CAM 3.0). Technical Report NCAR/TN-464+STR, National Center for Atmospheric Research, Boulder, Colorado.
- Costa, A., Macedonio, G., Folch, A., 2006. A three-dimensional Eulerian model for transport and deposition of volcanic ashes. *Earth Planet. Sci. Lett.*, 241 (34), 634–647.
- Connor, C.B., B.E. Hill, B. Winfrey, N.M. Franklin, and P.C. LaFemina, 2001, Estimation of volcanic hazards from tephra fallout, *Natural Hazards Review*, 2, 33–42.
- Dellino, P., D. Mele, R. Bonasia, G. Braia, L. La Volpe, R. Sulpizio, 2005. The analysis of the influence of pumice shape on its terminal velocity, *Geophys. Res. Lett.*, 32, L21306.
- Dutton, J., Fichtl, G., 1969. Approximate equations of motion for gases and liquids. *J. Atmos. Sci.* 26, 241– 254.
- Ganser, G., 1993. A rational approach to drag prediction spherical and nonspherical particles. *Powder Technology* 77, 143–152.
- Jacobson, M., 1999. Fundamentals of atmospheric modelling, 1st Edition. Cambridge University Press, New York.
- Macedonio, G., Costa, A., Longo, A., 2005. A computer model for volcanic ash fallout and assessment of subsequent hazard. *Computer and Geosciences* 31, 837–845.
- Pfeiffer, T., Costa, A., Macedonio, G., 2005. A model for the numerical simulation of tephra fall deposits. *J. Volcanol. Geotherm. Res.* 140, 273–294.
- Pielke, R., Cotton, W., Walko, R., Tremback, C., Nicholls, M., Moran, M., Wesley, D., Lee, T., Copeland, J., 1992. A comprehensive meteorological modeling system-RAMS. *Meteor. Atmos. Phys.* 49, 69–91.
- Small, C., Naumann, T., 2001. The global distribution of human population and recent volcanism. *Environ. Haz.* 3, 93–109.
- Ulke, A., 2000. New turbulent parameterization for a dispersion model in atmospheric boundary layer. *Atmos. Environ.* 34, 1029–1042.

- Wilson, L., and T. C. Huang, 1979. The influence of shape on the atmospheric settling velocity of volcanic ash particles, *Earth Planet. Sci. Lett.* 44, 311–324.
- Scire, J., Robe, F., Yamartino, R., 2000. A User's Guide for the CALMET Meteorological Model. Tech. Rep. Version 5, Earth Tech, Inc., 196 Baker Avenue, Concord, MA01742.
- Suzuki, T., 1983. A theoretical model for dispersion of tephra. In: D. Shimozuru, I. Yokoyama (Eds.), *Arc Volcanism: Physics and Tectonics*, Terra Scientific Publishing Company (TER-RAPUB), Tokyo.