Operational Multiscale Environment model with Grid Adaptivity

(OMEGA)

File Formats

Center for Atmospheric Physics
Science Applications International Corporation
1710 SAIC Dr., McLean, VA 22102.

Prepared by

Ananthakrishna Sarma

29 January 2003

Edition 2

© Copyright 2002, Science Applications International Corporation

Table of Figures

Figure 1.1: The OMEGA grid coordinate system with the grid layout (left) and a typical OMEGA grid element (right). 6
Figure 1.2: A typical OMEGA grid with the grid components detailed in the right side zoomed in view. Note that each edge has an implied direction from its first vertex to its ending vertex, which is indicated by the arrows. 7
Figure 3.1: Schematic of the layout of a PKB file. 11
Figure 3.2: Sample of a PKB trailer. 12

Table of Tables

Table 3.1: Fields written to the OMEGA output files. 16
Table 3.2: Default Parameters (written for both *puff* and *lpm* diffusion model output): 17
Table 3.3: Additional Parameters (written only for *puff* diffusion model output): 17
Table 3.4: Additional Parameters (written only for *lpm* diffusion model output): 17

Preface

In this document the following conventions are used to enhance clarity. File names, path names, environment variables etc., are displayed using a `fixed` (Courier New) font. If it is a variable that can take on different values at different times, such as command line options, it will be *`italicized`*. Names of executables, utilities and commands will be in **`bold-face`**. User provided input, either to the operating system or to one of the program components discussed in this document will be shown in blue color. For example to run a program called "myprog" one would enter

```
myprog -option1 -option2 … arg1 arg2 …
```

1. OMEGA Grid

OMEGA is based on an unstructured triangular prism grid, which is referenced to a rotating Cartesian coordinate system (Figure 1.1). The projection of the three-dimensional grid onto the reference sphere is made up of a mesh of triangular cells. The grid is unstructured as two neighboring cells need not be adjacent in the arrays that define and refer to those cells. The computational domain is bounded by one layer of boundary cells, which are assumed to be reflections of the cells just interior to the domain about the corresponding boundary. Figure 1.2 shows a grid with the bounding cells displayed.

1.1 Variable storage

The variables are stored at the cell centroids. However, the OMEGA grid is staggered in the vertical to improve computational accuracy. This means that the momentum variables are stored at different locations than the state variables. The state variables (air density, energy density as well as the concentration of all hydrometeors and aerosols) are stored at the cell centroids, while the momenta are stored at the center of the top faces (Figure 1.1, right panel). These face centers form the centroids of cells on a dual grid whose top bottom faces are half way in between the top and bottom faces of the cells on the primary grid.

Each individual cell of the grid (right panel of Figure 1.1) is a triangular prism, whose sides are vertical quadrilaterals (they lie along Earth's radii), and whose top and bottom faces are triangles. Near the surface of the Earth, the triangular faces are parallel to the surface beneath them and hence, are terrain-following. This restriction is gradually relaxed for cells at altitude and near the top of the domain the triangular faces are parallel to the reference sphere. Each cell is comprised of faces, edges and vertices. As the grid is structured in the vertical, the vertical faces can be referred directly to the edges of the cells as they form the projection of these faces onto the reference sphere.

Due to the unstructured nature of the grid, it is very important to save all the information on the grid connectivity so that the cell and its immediate neighbors can be easily

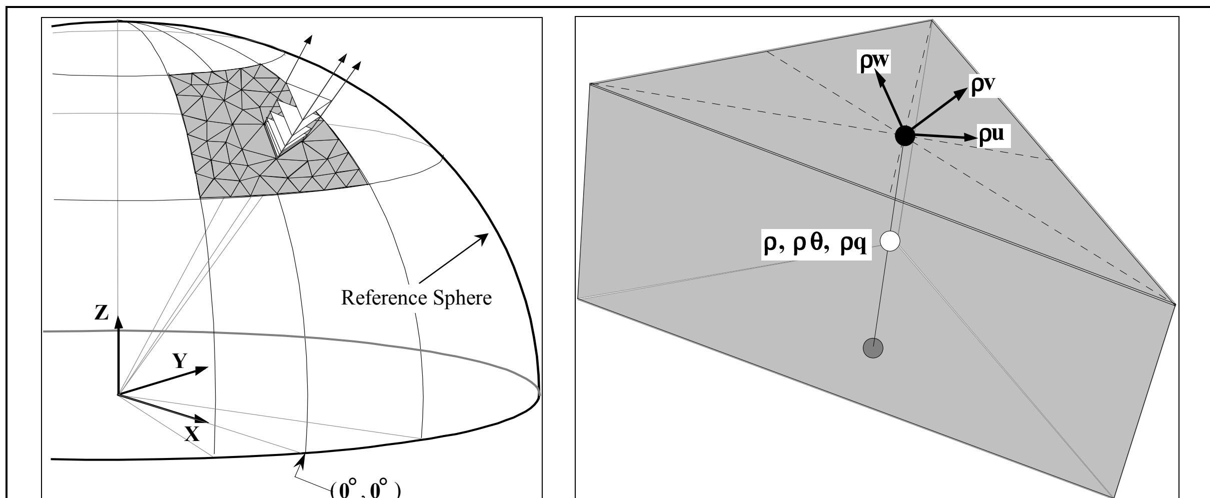

Figure 1.1: The OMEGA grid coordinate system with the grid layout (left) and a typical OMEGA grid element (right).

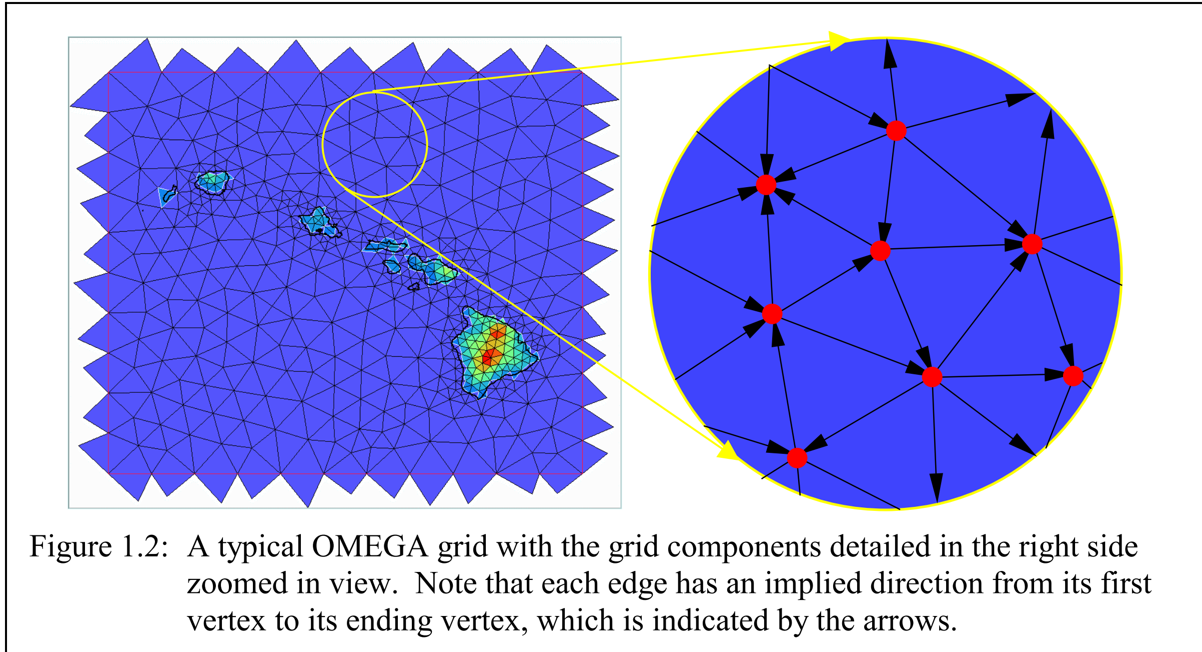

Figure 1.2: A typical OMEGA grid with the grid components detailed in the right side zoomed in view. Note that each edge has an implied direction from its first vertex to its ending vertex, which is indicated by the arrows.

accessed. This is facilitated by storing information at three basic levels – 1) vertex related information, 2) edge related information, and 3) cell related information.

1.2 Vertex Related Information

Vertices form the most basic elements of the OMEGA grid. The cells are defined by their edges, and the edges are in turn defined by their vertices. Vertices are characterized by their 3-Dimensional location. In the grid file the vertex locations are given by their latitude, longitude and altitude. Within OMEGA these are converted to the (x,y,z) locations on the Cartesian coordinate system. The vertices are numbered sequentially, as the grid generator creates each vertex. The vertex information also contains the index of one of the edges that contain the vertex, as well as a status flag which indicates whether the vertex is an interior or boundary vertex and whether the vertex represents an immovable point (such points are not allowed to be changed during adaptation) in the grid.

1.3 Edge Related Information

The edge related information in the grid file include the indices of the two vertices. The order in which these vertices are provided define an implicit direction for the edge. The edge is assumed to be directed from the first vertex to the second vertex. This directionality of the edges is important as it impacts the definition of the unit vectors, and the determination of velocity fluxes for advection. The vertex indices are followed by the indices of the two cells that flank that edge. – the cell to the "left" of the edge (left side determined as one looks from the first vertex of the edge to the second vertex) and the cell to the "right" of the edge. A flag is provided for each edge that indicates whether it is an interior or a boundary edge.

1.4 Cell Related Information

The cells are defined by the vertex and edge indices. The vertex indices are listed in a counter-clockwise order starting from an arbitrary vertex. The edges are then listed in the order edge connecting first vertex to the second, second to the third, and third to the first. If any of these edges have their vertices in the reverse order in the edge definition table, the edge index is provided as a negative number.

2. Grid File Format

The unstructured grid of OMEGA is defined by data contained in the grid file. In order to use the contents of the grid file effectively, it is necessary to understand the OMEGA grid structure and the features of a basic OMEGA grid element (Figure 1.1).

The grid file normally has an extension of `.grd` and is generated by the OMEGA grid generator. Grid files are also written by OMEGA as the grid is changed during dynamic adaptation. This is an ASCII file and contains information on the vertices, edges, cells and levels of the grid. The following provides the format of each record (record is a physical line that is terminated by a <CR> character) in this file.

Record 1: 6 floats – minimum longitude, maximum longitude, minimum latitude, maximum latitude, unused, radius of earth (m). These values are space-delimited.

Record 2: 10 integers – number of levels (NR), number of vertices (NV), number of edges (NE), number of cells (NC), number of boundary edges (NEB), maximum number of levels, maximum number of vertices, maximum number of edges, maximum number of cells, and maximum number of boundary edges. These values are space-delimited. (Older grid files will have only the first four values.

Record 3 – NV+2: Vertex information – NV records each containing: vertex number (integer), longitude of the vertex (float), latitude of the vertex (float), a vertex flag (integer), index of one edge that contains this vertex (integer). Each record is written in FORTRAN format (**`i6, 1x, 2f10.4, 2i6`**).

Record NV+3 – NV+2+NV×NR: NR×NV records containing a sequence number (integer), followed by the altitude (meters above MSL) of a vertex (float). The vertices at level 1 are given first, followed by those in level 2 etc. Each record is written in FORTRAN format (**`i6, 1x, f10.2`**).

Next NE records: Edge information – NE records each containing edge number (integer) followed by 5 integers which provide the indices of the beginning and ending vertices, and the left and right hand side cells and a status flag. The status flag indicates whether it is a boundary or internal edge. Each record is written in FORTRAN format (**`i6, 1x, 5i6`**).

Next NC records: Cell information – NC records each containing cell number (integer) followed by 6 integers which provide the vertex numbers of the first, second, and third (going counter-clockwise around the cell from an arbitrary vertex) vertices, and the edge numbers of the edges connecting the first vertex to the second, second to the third, and third to the fourth. Note from the edge-information array that there is a beginning and ending vertex for each edge. If in the cell definition, the order needs to be reversed, the edge number is provided as the corresponding negative number. Each record is written in FORTRAN format (**`i6, 1x, 6i6`**).

Next NC records: Land/water information – cell number followed by the land-water flag (either 0 or 1). ‘0’ indicates water and ‘1’ indicates land. Each record is written in FORTRAN format (**`i6, 1x, i2`**).

Next record: Boundary information flag. This is an ASCII string. If this string is equal to "boundaries" then the reading of the file will continue, otherwise the file will be closed and OMEGA will compute the boundary information. The following information will be missing from the grid produced by the OMEGA grid generator. It is only written by OMEGA when it writes a new grid during the dynamic adaptation sequence. This record is written in FORTRAN format (**a10**).

Next record: An integer providing the number of boundary cells. This record is written in FORTRAN format (**i6**).

Next NEB blocks: for $i = 1$ to NEB

First Record: Longitude and latitude of the ith boundary cell. This record is written in FORTRAN format (**2e20.12**).

Next NR+1 Records: (for j = 1 to NR+1)

The radial position and volume of the ith boundary cell at j^{th} level. Each of these records is written in FORTRAN format (**2e20.12**).

Next NR Records: (for j = 1 to NR)

The radial position and area of the top face of the ith boundary cell at j^{th} level. Each of these records is written in FORTRAN format (**2e20.12**).

Next record: x, y, and z position of the ith boundary cell centroid projected onto the reference sphere. This record is written in FORTRAN format (**3e20.12**).

Next Record: x, y, and z components of the radial unit vector through the ith boundary cell centroid projected onto the reference sphere. This record is written in FORTRAN format (**3e20.12**).

3. OMEGA Packed Binary Format Output

3.1 Introduction

A new format for OMEGA output was implemented in 1997 in order to avoid platform dependent files. In the past, data had been written in a blocked binary format, which restricted portability. The records and byte order for different platforms are highly dependent on the operating system and the compilers. Also, as more and more fields were included in these files with the inclusion of new physics modules, the size of the output files grew significantly. As the OMEGA variables are declared double precision, each floating-point value requires 8 bytes to store. However, post-processing programs do not require this level of precision. Hence, substantial savings in storage and I/O requirements can be attained by degrading the precision to lower but still acceptable levels.

A new type of database was designed which made the files platform-independent and thus portable. Additionally, a compression technique was introduced which reduces storage requirements but maintains accuracy at an acceptable, user-defined level. The packed binary database format uses an "object" type of access giving a high degree of flexibility in what is stored in the database. In the past, data files had to be read entirely into memory before one could access any of the data. The new format provides for faster access and lower memory requirements, because only the required data are read. Currently, the standard OMEGA output, the ADM particle output, and the momentum statistics data are written in this format.

3.2 OMEGA PKB File Format

The unstructured nature of the OMEGA computational grid makes it necessary to store grid related information in a file so that field data that are stored in the output files (referred to as PKB files) can be related to the appropriate grid cell. Post-processing routines that need to read OMEGA PKB files hence have to read the grid file first. Also, as the grid can change during the simulation due to dynamic grid adaptation, it is important that the appropriate grid is read.

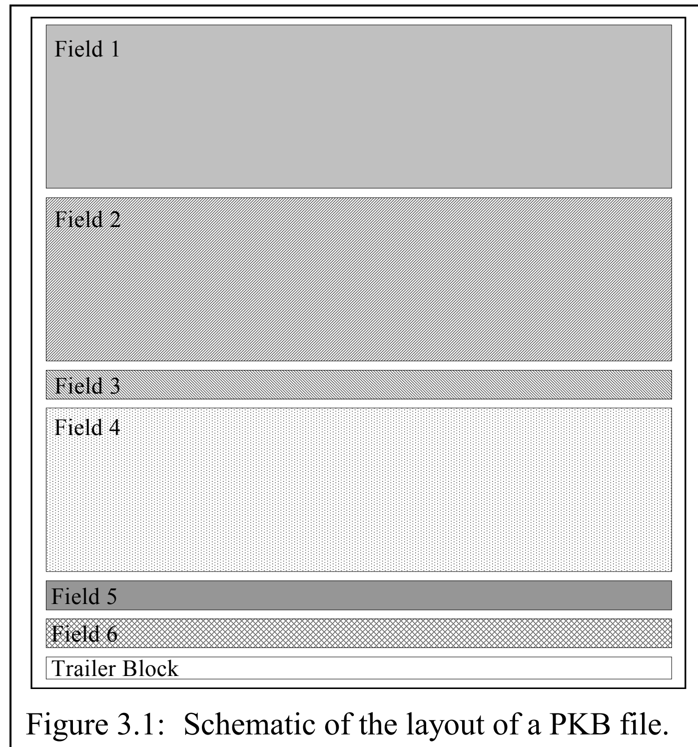

Figure 3.1: Schematic of the layout of a PKB file.

Each PKB file consists of two parts; 1) the packed binary data, and 2) the ASCII trailer table. A schematic of the PKB file layout is given in Figure 3.1. The packed binary data is made up of a sequence of fields or arrays written one after the other as indicated in Figure 3.1. Each of these fields is nothing but a byte stream.

The trailer table contains information about the complete data set followed by information about the individual fields within the file. The packed binary data is formatted according to field id, cell number, and level number. Each of these sections is described next.

3.2.1 ASCII Trailer format

One of the prime considerations in the design of the PKB format is speed of data access. The file is structured in such a way that any datum in the file can be accessed without having to read large sections of the file. This is achieved by creating a contents index and appending it as a trailer to the file. The trailer is written in ASCII format so that the file remains portable and so that it can be examined visually if needed. The trailer contains information on the OMEGA case, run identifier, grid file, other grid related information, the various fields output to the file, the beginning and ending byte-locations of these fields in the file, as well as the value ranges for each file.

A sample trailer from a PKB file is shown in Figure 3.2. The PKB file from which this trailer has been excerpted contains only two fields (indicated by NUMFLDS parameter), `uvel` and `vvel`. The last 12 bytes in the file contain the offset in bytes where the trailer table starts relative to the beginning of the file. Once this integer has been read, the file pointer can be moved to the beginning of the trailer using the *fseek* function and the entire trailer can then be read. The first six entries of the trailer contain information pertaining to the whole file. The first entry in each of these lines consists of a parameter name followed by its actual value. These parameters are described in the following tables.

```
#
# PKB TRAILER TABLE:
#
NBYTES      2
CASENAME    FLOYD
RUNID       1999091400
VALIDTIME   199909150300
GRDFILE     199909150300.grd
NUMFLDS     2
#
# FIELD INFORMATION:
#
# ID    START_POS    FLDTYPE    $UNITS$    $LABEL$
# IDIM    JDIM    NEB    MIN    MAX
#
uvel    0    0    $m/s$    $U Velocity$
5152    35    102    -6.84515531365928e+01    4.88322013313705e+01
#
vvel    367780    0    $m/s$    $V Velocity$
5152    35    102    -5.70147685299412e+01    6.22019616520333e+01
#
# TRAILER START POSITION:     735561
```

Figure 3.2: Sample of a PKB trailer.

NBYTES	int	The precision of the data in bytes. For some fields this value is overridden; if field type is 1 then 3 bytes are used for data values in that field.
CASENAME	char*	Identifies the simulation or domain.
RUNID	char*	Identifies the run, usually the start of the simulation
VALIDTIME	char*	yyyyMMddhhmm. The valid time of the data.
GRDFILE	char*	The name of the corresponding grid file.
NUMFLDS	int	The number of fields represented in the file.

Note: these entries are not required to be in this order, and additional entries are possible.

The rest of the trailer contains information on each of the fields represented within the file. This data, however, is required to be in the specified order given below. To preserve precision, the minimum and maximum data value information is written in double precision mode. Note also that all string entries such as the field label and units are entered with '$' as delimiters.

ID	char*	A unique string identifying the field.
START_POS	long int	The first position of the first byte of this field within the file.
FLDTYPE	int	If fldtype = 1, 3 bytes are used to represent each data value, otherwise NBYTES are used.
UNITS	char*	A string describing the units of output for this field, surrounded by $ as delimiters.
LABEL	char*	A string describing the field, surrounded by $ as delimiters.
IDIM	int	Horizontal dimension, i.e., number of cells.
JDIM	int	Vertical dimension, the number of levels.
MIN	double	The minimum data value of the field.
MAX	double	The maximum data value of the field.

With the information provided in the trailer, the position of any data value can be easily determined and the file pointer can be moved to extract that datum using the *fseek* function.

3.2.2 Packed Binary Data Format – the packing algorithm

In order to reduce the storage requirements for the OMEGA output files, the data values are compressed using a normalized compression algorithm. Each value is converted to an integer using the formula

$$I = \frac{F - F_{\min}}{F_{\max} - F_{\min}} N$$

where F is the data value, F_{max} and F_{min} are the maximum and minimum of the field (array), and N is a scale factor which determines the precision of I. For an n-byte precision, $N = 2^{8n} - 1 = 256^n - 1$. Thus the scaled values of the field will range from 0 to N. Then I is written to the file as a sequence of bytes with the least significant byte (LSB) written first. The following steps describe the procedure.

1. Determine how many bytes of precision are required (NBYTES).

2. Given an array of numbers, determine the maximum and minimum values within the array. MAX - MIN = RANGE.
3. Subtract the minimum value from each number in the array. Now we have a sequence of numbers ranging from 0 – RANGE. These numbers now require fewer bits to represent, as they are smaller.
4. Calculate a scaling factor:
   SCALE = (total range of available values given NBYTES) / RANGE
   SCALE = ((256 ^ N) - 1)) / RANGE
5. Multiply every value in the array by SCALE.
6. For each value in the array {
   For each byte of precision {
   write out as a byte (*value modulo 256*)
   reset *value = value / 256*
   }
   }

For example, converting a field to PKB format with 2-byte precision would entail the following steps.

1. Assume 2-byte precision.
2. We have an array (F) of numbers that range from 1000.5 – 1400.7.
   RANGE = 1400.5 – 1000.5 = 400.2
3. Subtract 1000.5 from all values. We now have an array ranging from 0 – 400.2 ($F – F_{min}$)
4. SCALE = ((256 ^ 2) - 1) / RANGE
   SCALE = 65535 / 400.2
   SCALE = 163.7556
5. Multiply all values in the array ($F – F_{min}$) by SCALE. We now have an integer array (I) ranging from 0 – 65,535.
6. If we have a value of 3000 (which by the way corresponds to an original data value of 1018.82) in this integer array.
   The first byte we write out is 3000 mod 256 = 184; this is the lowest order byte, or the least significant byte (LSB).
   Now we divide 3000 / 256 = 11 (we are dealing with integers, so the remainder is truncated).
   The second byte we write out is 11 mod 256 = 11.
   We now have a base 256 representation of our number.
   $\mathbf{3000} = (\mathbf{184} \times 256^0) + (\mathbf{11} \times 256^1)$.

The entry in the PKB file will be two bytes, the first one whose value will equal 184, and the second one whose value will equal 11.

3.2.3 Precision and Accuracy

Obviously, using just one byte of data precision will result in very low accuracy. Accuracy in the above example = 400.2 / 256 = 1.5633. However, accuracy increases

exponentially as number of bytes of precision increases. For example, if we used 2-byte precision in the last example, accuracy = 400.2 / 65,536 = 0.0061, which is accurate to the fourth significant figure. Accuracy increases linearly as range decreases. A range of 100 results in accuracy of 0.0015. If we use 3-bytes of precision, a range of 100 yields an accuracy of 2.33×10^{-8}, which is accurate to 10 significant figures. Since the IEEE standard for single precision floating-point numbers gives an accuracy of only 6 digits, using more than 2-byte precision is unnecessary for most applications. This is not the case if we are dealing with double precision numbers, which are accurate to 15 digits.

3.2.4 Unpacking the Data

As one would suspect, extracting values from a PKB file is exactly the opposite of the packing algorithm, except that we have much of the information already at hand. We have the maximum and minimum values for the field and we have the byte precision, all read from the trailer table. Thus we can easily compute the SCALE factor for that field. Given a location of the desired data, we can easily read the specified number of bytes and reconstruct the original number. The procedure is as follows.

1. Calculate the location of the data value in the file. Use the *fseek* function to move the file pointer to the first byte of the desired value.
2. Loop through the following:
   for (i=0; i<trailer.nbytes; i++) {
   order = pow(256, i) - 1;
   *intval = intval + fgetc(pkb_file) * order;*
   (where fgetc() gets the next byte from the PKB file.)
   }
   What this does is convert the base-256 representation to an integer (base-10) format.
3. Multiply this value by the SCALE factor.
4. Add the field MIN value to obtain the data value.

Note that if the field type is equal to 1 a 3-byte precision is used to represent the data values irrespective what the value of *trailer.nbytes* may be. A field-type of 1 is used in the ADM files, where large integers require precision higher than 2 bytes.

3.2.5 Data Layout in the PKB file

Each output field is usually an array within the OMEGA data structure. Some of the fields are two-dimensional (cells × levels, which actually makes them 3-dimensional in physical space), and others are 1-dimensional (either cells – 2-dimensional in space, or levels – 1-dimensional in space). Data values for the 2-dimensional fields are laid out as profiles. In other words, the data for levels 1[*] to NR (total number of levels) for cell 1 is written first, followed by those for cell 2, cell 3 etc. up to cell NC (total number of cells within the domain). Boundary cells are numbered from NC+1 to NC+NEB (NEB being the number of

[*] NOTE: Level one in OMEGA is a fictitious cell below the surface, which is rarely accessed. In OMEGA, the vertical levels range from level 2 – NR.

boundary cells). Therefore, if the boundary cell data is written out, it is at the end of the field data.

3.2.6 Contents of an OMEGA output PKB file

Even though there is no requirement as to which fields should be written to a PKB file, the following table shows the fields that are included in it by default. Several other fields may be available if the `yesupkb` switch is set to "`.true.`" in the **`omega.i`** file.

Table 3.1: Fields written to the OMEGA output files.

Field ID	Type	Label	Description	Units
uvel	real	U velocity	Zonal component of velocity	m/s
vvel	real	V velocity	Meridional component of velocity	m/s
wvel	real	W velocity	Vertical component of velocity	m/s
tpres	real	Pressure	Total pressure	mb
pres	real	Pressure Pert	Pressure Perturbation	mb
temp	real	Temperature	Temperature	°C
theta	real	Potential Temperature	Potential Temperature	K
rho	real	Air Density	Air Density	Kg/m^3
kappam	real	Kappam	Eddy diffusivity for momentum	
qvap	real	Vapor Concentration	Vapor Concentration	Kg/m^3
qdrop	real	Cloud Droplet Concentration	Cloud Droplet Concentration	Kg/m^3
qice	real	Cloud Ice Concentration	Cloud Ice Concentration	Kg/m^3
qrain	real	Rain Concentration	Rain Concentration	Kg/m^3
qsnow	real	Snow Concentration	Snow Concentration	Kg/m^3
cloud	real	Cloud Concentration	qdrop + qice	Kg/m^3
precip	real	Precipitation Concentration	qrain + qsnow	Kg/m^3
precip_gnd	real	Total Precip on Ground	Accumulated Precipitation	mm
clcov	real	Cloud Cover	Fractional Cloud Cover	
cumpreci	real	Cumulus Precipitation	Cumulus Precipitation	mm
cumcover	real	Cumulus cloud cover	Cumulus Cloud Cover	
qdust1	real	Dust-1 Concentration	Concentration of Dust Category 1	Kg/m^3
qdust2	real	Dust-2 Concentration	Concentration of Dust Category 2	Kg/m^3
…	…	…	…	…
qdust*N*	real	Dust-*N* Concentration	Concentration of Dust Category *N*	Kg/m^3
dust1_gnd	real	Dust-1 on ground	Accumulation on ground of Dust-1	mm
dust2_gnd	real	Dust-2 on ground	Accumulation on ground of Dust-2	mm
…	…	…	…	…
dust*N*_gnd	real	Dust-*N* on ground	Accumulation on ground of Dust-*N*	mm
tdust_gnd	real	Total dust on ground	Accumulation on ground of Dust	mm
hpbl	real	Boundary Layer Height	PBL Height	m
tground	real	Ground Temperature	Ground Skin Temperature	K
sst	real	OMEGA Sea Surface Temperature	Sea Surface Temperature	°C
bottm	real	Ocean bottom	Ocean Depth	m

3.2.7 Contents of an OMEGA ADM file in PKB format

Particle output files (.adm) contain time snapshots of particle locations and properties. The files are written at user-specified time intervals, usually every hour of model simulation time. The naming convention of the particle output files is: YYYYMMDDHHmm.adm,

where YYYYMMDD represents the date and HHmm the UTC time. The particle output can be written as ASCII-formatted or packed binary files. To write the particle output files as packed binary in OMEGA, the OMEGA model run setup file "omega.i" must set "`yesadmpkb = .true.`". Otherwise, the OMEGA particle output files will be written as ASCII files. The content of a particle output file, whether ASCII-formatted or packed binary, depends on which diffusion model was specified prior to the beginning of the simulation.

ASCII-formatted files use the same Fortran format for both *puff* and *lpm* diffusion model output; however, some of the quantities that are written to the file depend on which diffusion model was selected for the simulation. Packed binary files have a default set of particle information that is written to the file. The default information is followed by other particle information, which depends on the diffusion model that was selected for the simulation. The following tables show the fields that are written to the ADM packed binary file.

Table 3.2: Default Parameters (written for both *puff* and *lpm* diffusion model output):

Field ID	Type	Label	Description	Units
`iprt_td`	integer	ADM Output Time	Time the output file is written	s
`iprt_id`	integer	ADM Release ID	Release point ID number	
`iprt_pid`	integer	ADM Particle ID	Particle ID number	
`prt_lat`	real	ADM Latitude	Particle latitude	degrees
`prt_lon`	real	ADM Longitude	Particle longitude	degrees
`prt_alt`	real	ADM Altitude	Particle altitude	m-MSL
`prt_hav`	real	ADM Terrain Height	Ground altitude	m-MSL
`prt_t0`	real	ADM Injection Time	Particle injection time	s

Table 3.3: Additional Parameters (written only for *puff* diffusion model output):

Field ID	Type	Label	Description	Units
`prt_mss`	real	ADM Puff Mass	Mass of material in puff	kg
`prt_sx`	real	ADM Puff Sigma-X	Puff *x*-standard deviation	m
`prt_sy`	real	ADM Puff Sigma-Y	Puff *y*-standard deviation	m
`prt_sz`	real	ADM Puff Sigma-Z	Puff *z*-standard deviation	m

Table 3.4: Additional Parameters (written only for *lpm* diffusion model output):

Field ID	Type	Label	Description	Units
`prt_mss`	real	ADM Particle Mass	Mass of discrete particle	kg
`prt_rhd`	real	ADM Particle Density	Density of discrete particle	kg/m^3
`prt_d`	real	ADM Particle Diameter	Diameter of discrete particle	m

4. OMEGA Particle Output Files (.adm Files)

Particle output files (.adm) contain time snapshots of particle locations and properties. The files are written at user-specified time intervals, usually every hour of model simulation time. The use of particles during an OMEGA simulation is optional. To activate the Atmospheric Dispersion Model (ADM) in OMEGA that writes the particle output files, the OMEGA model run setup file "`omega.i`" must list: "*`yesadm = .true.`*".

The ADM in OMEGA can simulate the dispersion of material using two different diffusion models. One model (*puff*) assumes that each particle released is the centroid of a puff of uniformly distributed material whose volume increases over time. A second model (Lagrangian particle model – *lpm*) assumes that each particle released is a discrete mass of material. The choice of diffusion model for an OMEGA simulation is specified on the fourth line in the OMEGA model run setup file "`omega.adm`". If the line reads *`puff`*, the puff model will be activated. If the line reads *`lpdm`*, the lpm model will be activated.

The naming convention of the particle output files is: *`YYYYMMDDHHmm`*`.adm`, where *`YYYY`* represents the year, *`MM`* the month, *`DD`* the day, and *`HHmm`* the UTC time. The particle output can be written as either ASCII-formatted or packed binary files. To write the particle output files as a packed binary file, the OMEGA model run setup file "`omega.i`" must list: "*`yesadmpkb = .true.`*". Otherwise, the OMEGA particle output files will be written as ASCII-formatted files. The content of a particle output file, whether ASCII-formatted or packed binary, depends on which diffusion model was specified prior to the beginning of the simulation.

ASCII-formatted files use the same FORTRAN format for both *puff* and *lpm* diffusion model output; however, some of the quantities that are written to the file depend on which diffusion model was selected for the simulation. Packed binary files have a default set of particle information that is written to the file. The default information is followed by additional particle information, which depends on the diffusion model that was selected for the simulation.

4.1 Particle Output Packed Binary File Content and Formats

The contents of the ADM packed binary format file are listed in Table 4.1. The additional parameters included if the *puff* model is chosen are listed in Table 4.2, and those included if the *lpm* model is chosen are listed in Table 4.3. For a description of the PKB file format please refer to Chapter 3.

Table 4.1: Default Parameters (written for both *puff* and *lpm* diffusion model output):

Field ID	Type	Label	Description	Units
iprt_td	integer	ADM Output Time	Time output file is written	seconds
iprt_id	integer	ADM Release ID	Release point ID number	
iprt_pid	integer	ADM Particle ID	Particle ID number	
prt_lat	real	ADM Latitude	Particle latitude	degrees

prt_lon	real	ADM Longitude	Particle longitude	degrees
prt_alt	real	ADM Altitude	Particle altitude	m-MSL
prt_hav	real	ADM Terrain Height	Ground altitude	m-MSL
prt_t0	real	ADM Injection Time	Particle injection time	seconds

Table 4.2: Additional Parameters (written only for *puff* diffusion model output):

Field ID	**Type**	**Label**	**Description**	**Units**
prt_mss	real	ADM Puff Mass	Mass of material in puff	kg
prt_sx	real	ADM Puff Sigma-X	Puff x-standard deviation	m
prt_sy	real	ADM Puff Sigma-Y	Puff y-standard deviation	m
prt_sz	real	ADM Puff Sigma-Z	Puff z-standard deviation	m

Table 4.3: Additional Parameters (written only for *lpm* diffusion model output):

Field ID	**Type**	**Label**	**Description**	**Units**
prt_mss	real	ADM Particle Mass	Mass of discrete particle	kg
prt_rhd	real	ADM Particle Density	Density of discrete particle	kg/m^3
prt_d	real	ADM Particle Diameter	Diameter of discrete particle	m

4.2 Particle Output ASCII-Formatted File Content and Formats

The contents of the ASCII ADM output file and their format are given in Table 4.4.

Table 4.4: Contents of the ASCII ADM output file and their format

Description	**Units**	**Variable**	**Type**	**Format**
line 1: format: (2f10.2, 1x, i8, 1x, i7, 2(1x, i4.4, i8.8))				
Output time	seconds	time	real	f10.2
Dummy variable		dum	real	f10.2
Output iteration number		ntime	integer	i8
Number of particles (N_p)		numid	integer	i7
Simulation start date/time	YYYYMMDDHHmm	mdate0	integer	i4.4,i8.8
Simulation output date/time	YYYYMMDDHHmm	mdate	integer	i4.4,i8.8
Lines 2 through N_p+1: format (i5, 1x, i3, 1x, i7, 1x, f7.3, 1x, f8.3, 1x, f9.2, 1x, f8.0, 2(1x, f7.2), 1x, e10.3, 2(1x, f9.2), 1x, f9.2, 2(1x, e10.3))				
Release point ID number		iprt_id	integer	i5
Release level ID number		iprt_lv	integer	i3
Particle ID number		iprt_pid*ip	integer	i7

Particle latitude	degrees	prt_lat	real	f7.3
Particle longitude	degrees	prt_lon	real	f8.3
Particle altitude	m-MSL	prt_alt	real	f9.2
Particle release time	seconds	prt_t0	real	f8.0
Ground altitude	m-MSL	prt_hav	real	f7.2
Particle density	kg/m^3	prt_rhd	real	f7.2
Puff vapor concentration	kg/m^3	prt_vap	real	e10.3
Puff *x*-standard deviation	m	prt_sx	real	f9.2
Puff *y*-standard deviation	m	prt_sy	real	f9.2
Puff *z*-standard deviation	m	prt_sz	real	f9.2
Particle mass	kg	prt_mss	real	e10.3
Particle diameter	m	prt_d	real	e10.3

5. OMEGA Concentration Output File (.con) Content and Formats

OMEGA concentration output files (.con) contain time snapshots of the 3-D distribution of an effluent released during an OMEGA simulation. The files are written at user-specified time intervals, usually every hour of model simulation time. The calculation of concentrations during an OMEGA simulation is optional. To activate the concentration calculation routines, the OMEGA model run setup file "`omega.i`" must list: "*`yesconc = .true.`*" and "*`yesecon = .true.`*" followed by an output interval at which the files will be written, (e.g., "*`dtconc = 1.0`*").

The OMEGA concentrations are calculated on a dynamically adapting, structured, 3-D receptor grid that is overlaid on the OMEGA unstructured grid. The dimensions of the structured receptor grid are a function of particle latitude, longitude, altitude, and puff properties. The default number of equally spaced vertical levels at which concentrations are calculated is fixed at 11. Thus, the altitudes at which each of the 11 vertical levels is located may change over time and depends on the depth of the atmosphere through which the puffs range. The default number of latitude and longitude receptor grid points is fixed at 300 each.

The naming convention of concentration output files is: *`YYYYMMDDHHmm`*`.con` where *`YYYY`* represents the year, *`MM`* the month, *`DD`* the day, and *`HHmm`* the UTC time. The concentration file contains 4 sections. The first section provides the time and general grid information. This is followed by the latitude and longitude values of the concentration grid. Then for each vertical level of the concentration grid the altitude and concentration values are output. The FORTRAN code segment shown in Figure 5.1 depicts the structure of the concentration file.

```
      .
      .
      .
      real ylat(nx,ny), xlon(nx,ny), hgtmsl(nx,ny,nz)
      real hgtmsl(nx,ny,nz), conc(nx,ny,nz)

      read(iunit, 3002) mdate0, mdate, nylat, nxlon, nzalt
c --- Read the array of latitude points
      read(iunit,2000)((ylat(i,j),i=1,nx),j=1,ny)
c --- Read the array of longitude points
      read(iunit,2000)((xlon(i,j),i=1,nx),j=1,ny)
      do k = 1, nz
         read (iunit, 2000) ((hgtmsl(i,j,k), i=1,nx), j=1,ny)
         read (iunit, 2000) ((conc(i,j,k), i=1,nx), j=1,ny)
      enddo
 1000 format (a12, 1x, a12, 1x, i4, 1x, i4, 1x, i4)
 2000 format (8e15.7)
      .
      .
      .
```

Figure 5.1: FORTRAN code to read an OMEGA concentration file. The concentrations are defined on a `nx`×`ny`×`nz` structured grid. In this example the arrays `ylat` and `xlon` give the latitude and longitudes of the grid points while `hgtmsl` gives the altitude above MSL. The array `conc` holds the concentration values.

INDEX

Dynamic Adaptation 9
PKB.. 11
    ADM File 16
    Algorithm... 13
    Data Layout.................................. 11, 15
    Data Precison 13, 14
    Output File 15
    Trailer Table 11
Unstructured grid..................................6, 9
    Boundary Cells...................................10
    Cells...9
    Dual Grid...6
    Edges 9
    Levels 9
    Staggered Grid......................................6
    Vertices...9