MUFITS

Multiphase Filtration Transport Simulator
version 2013.B

Files Format

Author: Andrey Afanasyev

Date: 10 may 2013

Moscow, Russia

Contents

1. The database format **3**

1.1. File types . . . 3

1.2. The structure elements of the database . . . 4

1.3. Record format . . . 4

1.4. Block format . . . 5

1.5. Mandatory records . . . 7

2. SUM-file format **8**

2.1. Record TIME . . . 8

2.2. Record DATE . . . 9

2.3. Names of the blocks . . . 9

2.4. Record ARRAYS . . . 10

2.5. Record DATA . . . 14

3. MVS-file format **15**

3.1. Names of the blocks . . . 15

3.2. Record GRIDSIZE . . . 15

3.3. Record POINTS . . . 16

3.4. Record CELLS . . . 16

1 The database format

1.1. File types

The MUFITS hydrodynamic module creates the files of two types which have identical internal structure of the database. These files have different extensions .SUM and .MVS because of the different purpose of the files. Farther, we reference to the files of these types as SUM-file and MVS-file.

File types		
SUM-file	-	The file is used to store the results of the simulation. The file can contain the time or date value and the properties (e.g. pressure, temperature, permeability, fluxes, injection rates) for cells, connections, sources, reservoir regions, etc.
MVS-file	-	The file is used to store the geometrical properties of the reservoir for subsequent visualization purposes. The data items stored in the file are not changing during the simulation. The file can contain the vertex coordinates of cells, well trajectories, point source coordinates, etc.

The file of each type can be represented (saved) in two modes.

Data storage mode		
Formatted file	-	The file is based on a formatted representation of data. This file can be read and edited in any text editor.
Binary file	-	The file is based on a binary representation of data. This file can't be read by text editors.

The sequence of data items in each mode is identical. Therefore farther we describe the MUFITS files format for the two modes simultaneously.

1.2. The structure elements of the database

Every file consists of an arbitrary number of data records. Father we consider the data records of two types which we reference to as "record" and "block".

Data record types		
Record	-	"Record" is a simple data record which means that the record contains only data items and doesn't contain nested records.
Block	-	"Block" is a composed data record. The record is composed of nested "records" and "blocks". The "block" doesn't directly contain data items.

The blocks and the files themselves created by MUFITS hydrodynamic module are composed of an arbitrary sequence of blocks and records (Fig. 1.1). The file (database) itself can be considered as a main parent block.

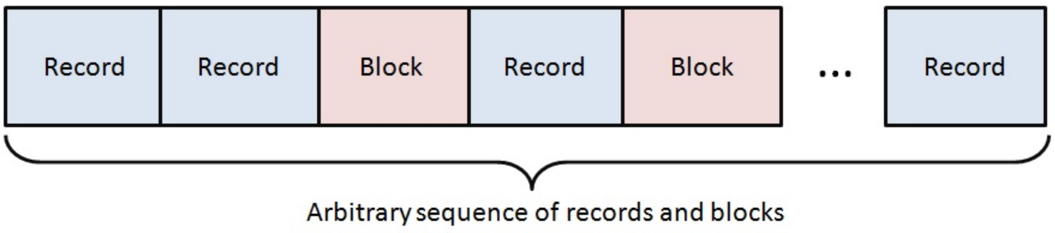

Figure 1.1: The file structure

1.3. Record format

Binary file

The structure of the record in binary mode is demonstrated in Fig. 1.2. The record begins with prefix of the record name and the record size. The name is 8 byte character. Therefore the records name must be composed of no more than 8 letters. All the letters must be capital. If the name is composed of less than 8 letters then the remaining positions in the record name data item must be blank.

The second data item is 8 byte integer for the size of the record body in bytes. The sizes of all remaining data items in the record body must sum to this data item.

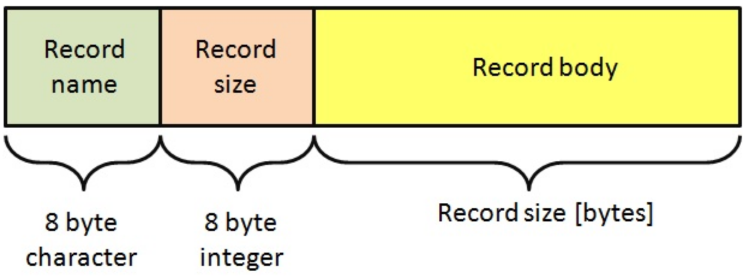

Figure 1.2: Record synatax in binary file

After the prefix, the record contains the data items (the record body) which are specific for every individual record indentified by the name.

The record is empty if it doesn't have associated data items. In this case the record length is equal to 0.

Formatted file

The syntax of the record in formatted mode is given below.

Record syntax in formatted file

```
'Record name'
    'Record body'
/
```

All definitions are the same as in the description of binary mode. The record name must be positioned in a new line of the formatted file. The blank spaces before the record name are prohibited. Only the first 8 letters in the record name are significant. If the record name has less than 8 letters then the remaining blanks are not required. The record length is not saved in the formatted mode. The record body which contains the data items must be positioned in a new line immediately after the record name. The data items in the record body must be delimited by an arbitrary (greater than 0) number of blank spaces. The blank spaces can also be before the record body. The parts of the record body can be carried on to the next line. The record must be terminated by the slash sign which must be positioned in a new line after the record body.

An empty record is followed by the slash sign in the next line to the line of the record name.

1.4. Block format

Binary file

The structure of the block in binary mode is demonstrated in Fig. 1.3. The block begins with prefix of the block name and the block size. The name is 8 byte character. Therefore the block name must be composed of no more than 8 letters. All the letters must be capital. If the name is composed of less than 8 letters then the remaining positions in the block name data item must be blank.

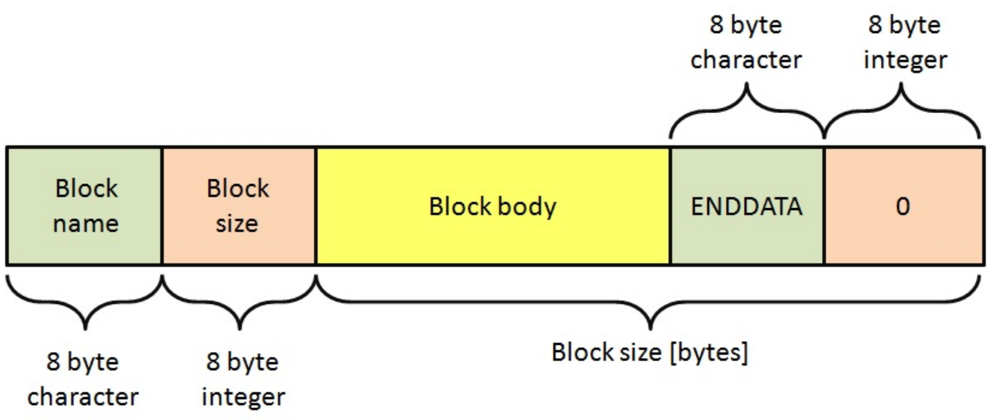

Figure 1.3: Block synatax in binary file

The second data item is 8 byte integer for the size of the block body in bytes. The sizes of all remaining data items and the block postfix (see below) in the block body must sum to this data item.

After the prefix, the block contains the nested blocks and records which are specific for every individual block indentified by the name.

The block must be terminated an empty record ENDDATA which is the postfix of the block. The postfix consists of 8 byte character ENDDATA (the last position is blank) and 8 byte integer having value 0 (because the record ENDDATA is empty).

The prefix and postfix of the block are the brackets for the nested records and blocks.

Formatted file

The structure of the record in the formatted mode is given below.

Block syntax in formatted file

```
'Block name'
      'Block body'
ENDDATA
/
```

All definitions are the same as in the description of binary mode. The block name must be positioned in a new line of the formatted file. The blank spaces before the block name are prohibited. Only the first 8 letters in the block name are significant. If the block name has less than 8 letters then the remaining blanks are not required. The block length is not saved in the formatted mode. The nested records and block must be positioned after the line with the block name. Every new record and block begins in a new line. The records and blocks can be delimited by an arbitrary number of empty lines. The block is terminated by the empty record ENDDATA.

1.5. Mandatory records

Binary file

The empty record BINARY must be the first record of the file in binary mode. The empty record ENDFILE must be the last record of the file (Fig. 1.4).

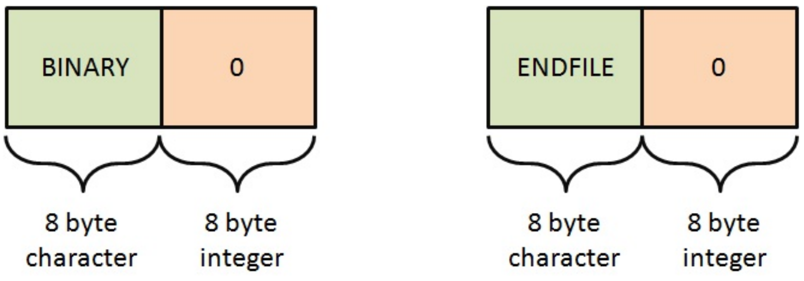

Figure 1.4: Mandatory records syntax in binary file

Formatted file

The empty record ASCII must be the first record of the file in formatted mode. The empty record ENDFILE must be the last record of the file.

Mandatory records in formatted file

```
ASCII
/

ENDFILE
/
```

2 SUM-file format

2.1. Record TIME

The record TIME specifies the time in the simulation schedule. The following data in the SUM-file are for this time in the schedule until time is redefined by a new record TIME. The record time must be before any block of simulation data.

The syntax of the record TIME is demonstrated in Fig. 2.1 for binary SUM-file and in the text below for formatted SUM-file.

Record TIME syntax in formatted file

```
TIME
    'Time value'  'Dimension'
/
```

The record body contains two data items:

1. The time value (8 byte real);
2. The dimension of the time value (8 byte character). All letters must be capital. By default this data item is DAYS.

The record length is 16 bytes in binary file.

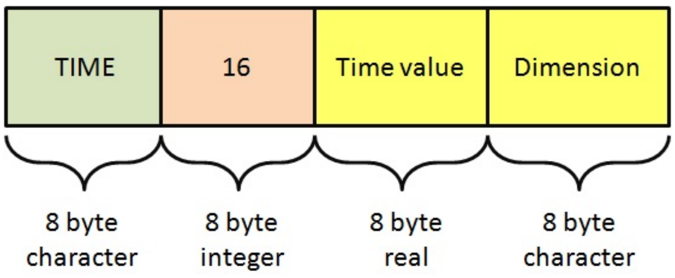

Figure 2.1: Record TIME syntax in binary file

2.2. Record DATE

The record DATE specifies the date in the simulation schedule.

The syntax of the record DATE is demonstrated in Fig. 2.3 for binary file and in the text below for formatted file.

Record DATE syntax in formatted file

```
DATE
   'Day'   'Month'   'Year'
/
```

The record body contains three data items:

1. Day (4 byte integer);
2. Month (8 byte character) (allowed values JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC; the remaining positions are blank);
3. Year (4 byte integer).

The record length is 16 bytes in binary file.

The data provided in the record DATE are supplementary for the data provided in the record TIME.

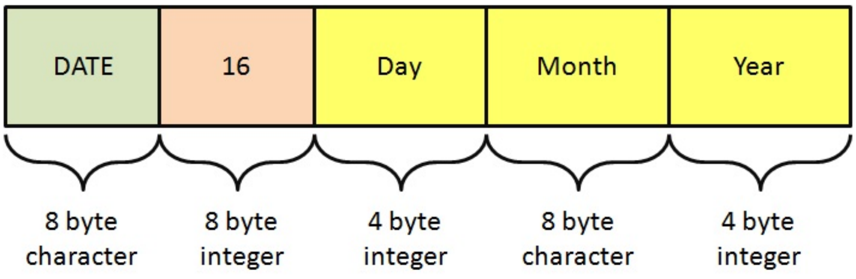

Figure 2.2: Record DATE syntax in binary file

2.3. Names of the blocks

The following blocks can be created by MUFITS in the SUM-file.

List of SUM-file blocks

Block name	Object	Example of the saved property	1st property

CELLDATA	Cell	Pressure (PRES), saturation (SAT), porosity (PORO), etc.	CELLID
CONNDATA	Logical connection between cells	Transmissibility (TRAN), flux (FLUX1), etc.	CONNID
SRCDATA	Point source	Mode (SRCMODE), cumulative flux (CUMFLUX1), etc.	SRCID
FPCEDATA	FIPNUM region	Average pressure (PRES), mass-in-place (MC1#NAME), etc.	FIPCELL
FPCODATA	Boundary between FIPNUM regions	Flux (FLUX1), cumulative flux (CUMFLUX1), etc	FIPCONN

The names of the blocks allow distinguish the type of the data stored in the block before reading the body of the block. We introduce the following notation:

- Object - an element of numeric model, which can be cell, connection, etc.;
- Property - a parameter of numerical model, which can be porosity, flux, etc. The property is specified by object type. For example, porosity is defined for cell but not for connection or point source object (see table above).

The table above defines the properties of which exactly objects are stored in the blocks. Every block must include the flowing records.

List of nested records		
ARRAYS	-	The header for the data in the record DATA.
DATA	-	The data as specified by the header ARRAYS.

In every block, the records must be in the sequence they are given in the table above.

2.4. Record ARRAYS

Syntax

The record ARRAYS describes the sequence of the property values in the current block. This record is the header for the subsequent record DATA. The syntax of the record ARRAYS is demonstrated in Fig. 2.3 for binary file and in the text below for formatted file.

Record ARRAYS syntax in formatted file
```
ARRAYS
    'Number of properties'   'Number of objects' /
    'Mnemonic' 'Property dimension' 'Tag' ... 'Tag' /
    ...
    'Mnemonic' 'Property dimension' 'Tag' ... 'Tag' /
/
```

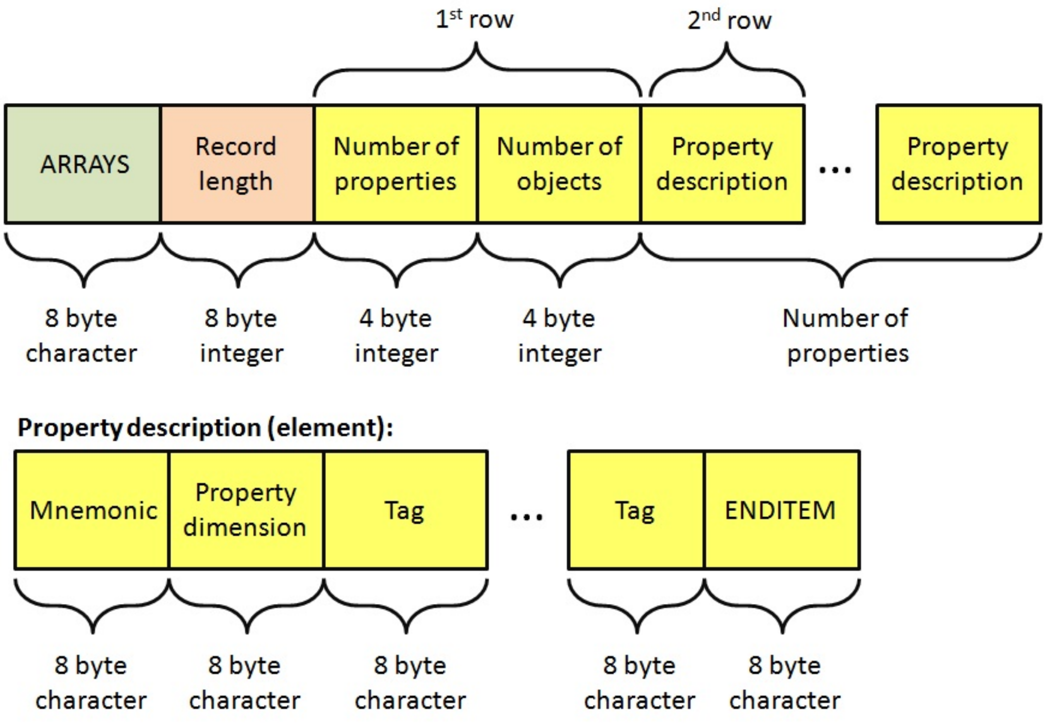

Figure 2.3: Record ARRAYS syntax in binary file

The record is followed by a list with arbitrary number elements. The length of the first element is fixed (8 bytes; two data items 'Number of properties' 'Number of data objects') while the other elements have an arbitrary length.

In formatted file, every new element begins on a new line and terminates by the slash sign. In binary file, the first element doesn't have the termination data item because this element has fixed length. The other elements are terminated by the 8 byte character ENDITEM (the last position is blank).

The first element in the list of the record ARRAYS contains two data items:

1. The number of properties saved in the record DATA for every object (4 byte integer). This is the number of subsequent elements in the record ARRAYS.

2. The number of objects for which the data are provided in the record DATA. This is the number of the list elements in the record DATA.

Every of the following elements in the record ARRAYS contains the description of the related property:

1. The first data item in the element is the property name - mnemonic (8 byte character) (review the document List of Mnemonics);
2. The second data item is the property dimension (8 byte character). If this value is NODIM then the property is dimensionless. If the value is SI then the property is in SI units;
3. The following data items in the element can be the property tags which define the property in details (8 byte characters). The number of tags is arbitrary;
4. Eevery element is terminated by 8 byte character ENDITEM.

The first property specified in the second element of the record ARRAY must be the object ID. For example, the first property in the block CELLDATA must be the cell ID which is the property CELLID (see the List of SUM-file blocks).

Property tags

There are three types of the property tags:

1. Data type tags;
2. Output mode tags;
3. Phase state tags.

Every property must be associated with only one tag of each type. The data type tags specify the type of the data items in the record DATA associated with the property. By default (if no tag of this type specified) the data items are 8 byte real numbers (the property is associated with the tag REAL8).

Data type tags		
INT1	-	1 byte integer
INT2	-	2 byte integer
INT4	-	4 byte integer

REAL4	-	4 byte real number
REAL8	-	8 byte real number (default)
CHAR4	-	4 byte character
CHAR8	-	8 byte character

The output mode tags are given in the table below. If the property is associated with the tag SINGLE then a single value of the property is provided for every object in the record DATA (e.g. porosity value for every cell). If the property is associated with the tag DOUBLE then two values of property are sequentially provided in the record DATA (e.g. the distances from cell center to cell face for two connected cells). The tag DOUBLE is usually associated with properties defined for logical connections. By default the property is associated with the tag SINGLE.

Output mode tags		
SINGLE	-	A single property is output, e.g. a property in a cell (default)
DOUBLE	-	Two properties are output, e.g. properties for each of two connected cells

The phase state tags are given in the table below. If the property is associated with the tag STATE0 then a single value of the property is provided in the record DATA for every object regardless the phase state of the mixture (the property PHST). If the property is associated with the tag STATE1 then the number of the property values provided for every object in the record ARRAY is equal to the value of the property PHST. If a property is associated with the tag STATE1 then the property PHST must be also specified in the record ARRAY. The property PHST must be specified before any property associated with the tag STATE1.

Phase state tags		
STATE0	-	A single property is output. The property PHST doesn't affect the output (default)
STATE1	-	The number of output properties is equal to the value of the property PHST

In formatted file, three data items are always provided in the record DATA for the property associated with the tag STATE1 but only the first 'PHST value' data items are relevant. The remaining data items must be ' 1* '. In binary file, the number of data items for the property associated with the tag STATE1 is always equal to the value of the property PHST.

2.5. Record DATA

The record DATA contains the sequence of the property values which are specified in the current block. The specification of the data items sequence in this record is provided in the record ARRAYS. The syntax of the record DATA is demonstrated in Fig. 2.4 for binary file and in the text below for formatted file.

Record DATA syntax in formatted file

```
DATA
   'Object properties' /
   ...
   'Object properties' /
/
```

The record DATA is a list of the elements 'Object properties'. Every element of the list contains the data for every individual object. First element includes all the properties specified by the record ARRAYS for the first object. Second element includes all the properties for the second object, etc. For every object the data items are provided in the sequence the properties are specified in the record ARRAYS.

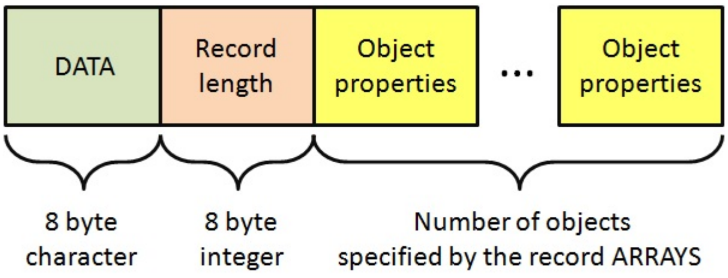

Figure 2.4: Record ARRAYS syntax in binary file

In the binary file, the elements 'Object properties' are not delimited. In the formatted file, every element 'Object properties' begins on a new line and is terminated by the slash sign. The number of the elements 'Object properties' is specified in the record ARRAYS.

Any record DATA must be preceded by the record ARRAYS in the same block.

3 MVS-file format

3.1. Names of the blocks

Currently, MVS-file can include only one block GRIDDATA which specifies the geometrical parameters of the computational grid. By default the grid is composed of hexahedral cells. Every block GRIDDATA must include the following records

List of nested records		
GRIDSIZE	-	Defines the number of cells and cell vertexes.
POINTS	-	Defines the cell vertexes coordinates.
CELLS	-	Defines the vertexes of every cell.

In the block GRIDDATA, the records must be in the sequence they are shown in the table above.

3.2. Record GRIDSIZE

The syntax of the record GRIDSIZE is demonstrated in Fig. 3.1 for binary file and in the text below for formatted file.

Record GRIDSIZE syntax in formatted file

```
GRIDSIZE
    'Number of vertexes'   'Number of cells'
/
```

The record body contains two data items:

1. The number of vertexes for all grid cells (4 byte integer). This is the number of elements in the subsequent record POINTS;
2. The number of grid cells (4 byte integer). This is the number of elements in the subsequent record CELLS.

The record length is 8 bytes in binary file.

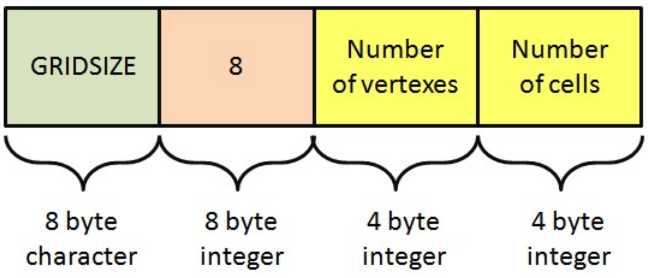

Figure 3.1: Record GRIDSIZE syntax in binary file

3.3. Record POINTS

The syntax of the record POINTS is demonstrated in Fig. 3.2 for binary file and in the text below for formatted file.

Record POINTS syntax in formatted file

```
POINTS
      'x-coordinate'   'y-coordinate'  'z-coordinate' /
       ...
      'x-coordinate'   'y-coordinate'  'z-coordinate' /
/
```

The record POINTS is a list. Every element of the list contains three data items which are the coordinates of a vertex:

1. The first data item is the x-coordinate of the vertex (8 byte real number);
2. The second data item is the y-coordinate of the vertex (8 byte real number);
3. The third data item is the z-coordinate (depth) of the vertex (8 byte real number).

By default all coordinates are given in meters.

The number of the elements in the list is equal to the number of vertexes specified in the record GRIDSIZE. In binary file, the elements of the list are not delimited. In formatted file, every new element begins on a new line and terminates by the slash sign.

3.4. Record CELLS

The syntax of the record CELLS is demonstrated in Fig. 3.3 for binary file and in the text below for formatted file.

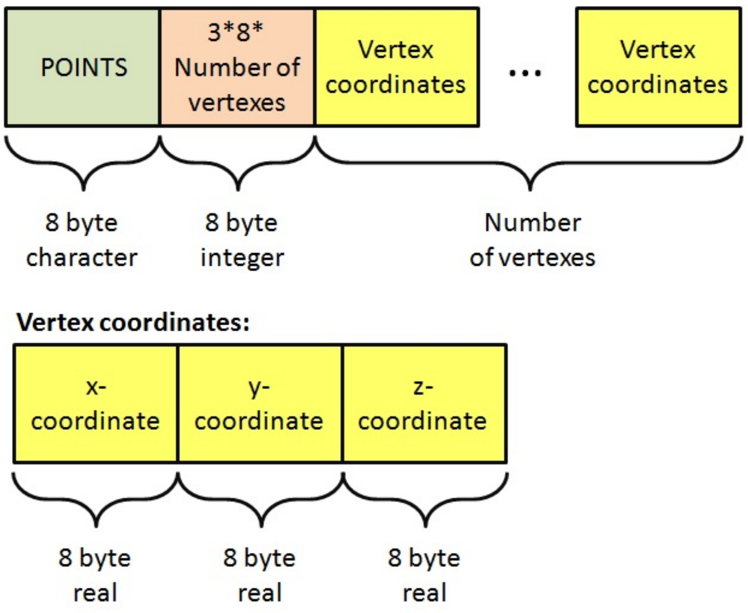

Figure 3.2: Record POINTS syntax in binary file

Record CELLS syntax in formatted file

```
CELLS
      'Cell ID' 'P#(0,0,0)'   'P#(1,0,0)'  'P#(1,1,0)'  'P#(0,1,0)'
                'P#(0,0,1)'   'P#(1,0,1)'  'P#(1,1,1)'  'P#(0,1,1)'  /
     ...
      'Cell ID' 'P#(0,0,0)'   'P#(1,0,0)'  'P#(1,1,0)'  'P#(0,1,0)'
                'P#(0,0,1)'   'P#(1,0,1)'  'P#(1,1,1)'  'P#(0,1,1)'  /
/
```

The record POINTS is a list. Every element of the list contains nine data items which are as follows:

1. The first data item is the cell ID (4 byte integer). This is the property CELLID;
2. The following 8 data items are the numbers of vertexes in the list specified by the record POINTS (4 byte integers). These 8 data items are the vertexes of the corresponding hexahedral cell identified by the ID (the property CELLID). The vertexes ordering rule for every cell can be seen in Fig. 3.3 and 3.4.

The number of the elements in the list is equal to the number of cells specified in the record GRIDSIZE. In binary file, the elements of the list are not delimited. In formatted file, every new element begins on a new line and terminates by the slash sign.

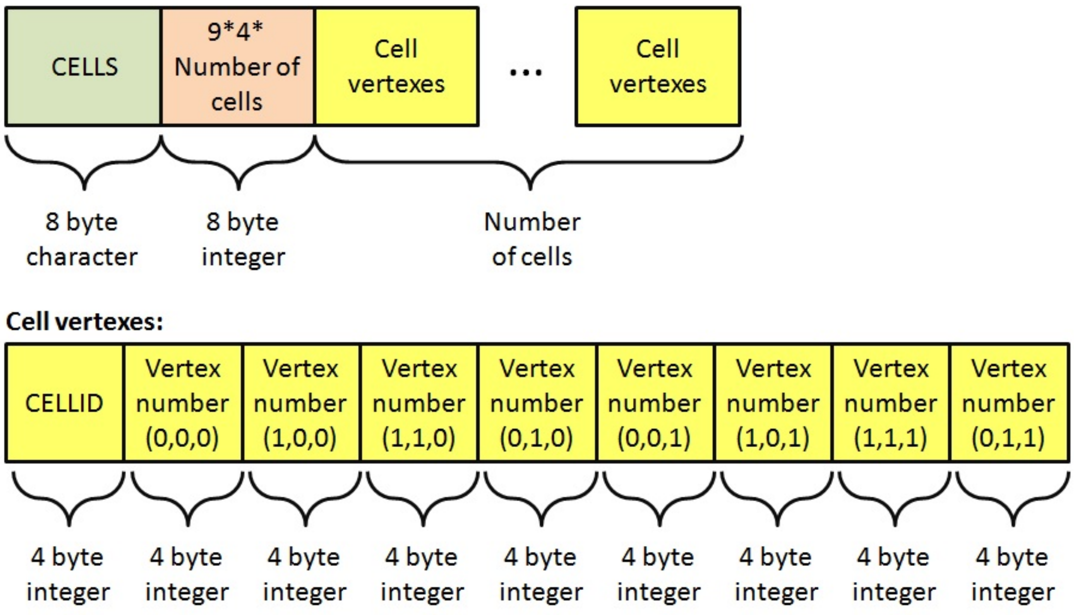

Figure 3.3: Record CELLS syntax in binary file

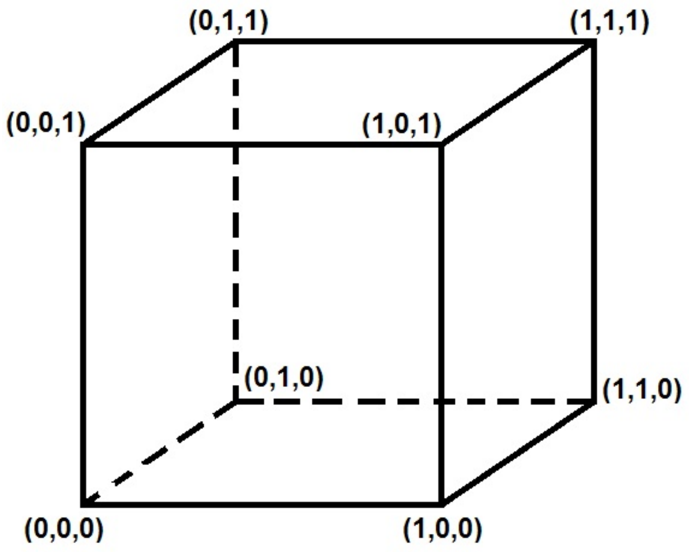

Figure 3.4: Vertexes numbering