MCTAL API reference manual

N. Borghi

Abstract

MCNPX is one of the most robust and diffused tool for Monte Carlo simulations in neutron physics. The results of MCNPX simulations can be saved in different formats and one of the most common is the ASCII format of the MCTAL file. In this article, an API to the MCTAL file is described.

Contents

1 Introduction

Monte Carlo simulations and neutron physics are two deeply interdependent subjects of study. The fast growth of neutron physics, both theoretical and experimental, would not have been possible without the support of Monte Carlo simulations. MCNPX (Monte Carlo N-Particle eXtended) is certainly the most significant Monte Carlo software currently available to neutron scientists and its history goes hand in hand with the history of neutron physics. MCNPX provides the users with a huge set of tools to study all the aspects of neutron interactions with matter and it is also able to deal with all particles at all energies.

This work aims to deal with the MCNPX output data, which can be delivered to the user in different formats, both ASCII and binary. One of the most common output formats of MCNPX is the plain-text *mctal* file, on which this work is focusing.

The information available from a simulation performed with MCNPX is remarkably complex and, when saved to an ASCII file, makes the file itself extremely dense and completely human unreadable. MCNPX users are continuously developing tools for reading their own mctal files for extracting just the needed information. The aim of this code is to provide a general tool for reading every mctal file, in all the possible configurations and to store its information in a class to which users can interface and convert the data in their favourite format for analysis.

This API is part of a bigger project started by Dr. Batkov and named Mc-Tools. The entire source code is available at https://code.google.com/p/mc-tools/ .

2 Structure of the mctal file

The structure of a typical mctal file is reasonably simple and it can be divided in two main blocks:

- **Header**: it contains a summary of the MCNPX settings used when the simulation was run and a brief overview of the tallies contained in the mctal file.
- **Tally**: for each tally requested in the input file, there is a block of data starting with the word *tally* and ending with the list of *tfc* (Tally Fluctuation Chart) values.

Information contained both in headers and in tallies is very dense and it is important to give an overview of what is contained in these lines [1, 2, 3, 4] in order to understand the importance of this conversion tool.

2.1 Header

In Code 1 the header of a typical mctal file is shown.

```
mcnpx       2.5.0  10/10/10 05:13:49    2   30000000   600197072467
c ESS example comment line of MCNPX simulation
ntal    14
     1    4    5    6   11   14   15   16   21   24   26   31   36   41
```

Code 1. *Header of the mctal file.*

This is the structure of the data:

- Line 1:
  - Name of the code[1]
  - Version of the code[1]

[1]Could also be omitted.

  - Date and time when the simulation was run^1
  - Dump number
  - Number of histories that were run
  - Number of pseudorandom numbers used

- Line 2:
  - Problem identification line, which corresponds to the first line of the input file
- Line 3:
  - Keyword *ntal* followed by the number of tallies contained in the mctal file
  - Keyword *npert* followed by the number of perturbations in the problem^1
- Line 4:
  - List of the tally numbers on as many lines as necessary

2.2 Tally

The tally block, shown in Code 2, begins with the keyword *tally* and ends with the beginning of a new tally, if present, or with the end of the mctal file. Each tally block contains the following information:

- Line 1:
  - Keyword *tally* followed by: the tally number, a number indicating the particle type (if > 0) or the number of particle types (if < 0) and a number indicating the type of the detector tally
- Line 2:
  - List of 0/1 indicating the particle types used by the tally
- Line 3:
  - Tally comment line(s)^1
- Lines 4 - 13:
  - Cards `f`, `d`, `u`, `s`, `m`, `c`, `e`, `t` (respectively cell, total vs. direct or flagged vs. unflagged, user, segment, multiplier, cosine, energy and time bins) are listed one per line interleaved with the corresponding detailed information. Moreover, `u`, `s`, `m`, `c`, `e`, `t` can also be followed by a `t` or `c` option whenever total or cumulative binnings are required by the user
- Line 14:
  - Keyword *vals* indicates the beginning of the values section
- Lines 15 - 40:
  - Values produced by the simulation followed by the corresponding relative error
- Line 41:
  - Keyword *tfc* opens the last section of the tally body, which contains the tally fluctuation data. It is followed by the number of sets of tally fluctuation data and by a list of eight numbers indicating the bin indexes of the tally fluctuation chart bin
- Lines 42 - 56:

```
tally    1   -1    0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
         Current tally for the upper target surface
f        1
    116
d        1
u        0
st      51
m        0
c        2
  0.00000E+00  1.00000E+00
e        0
t        0
vals
  9.32709E-01 0.0020  1.29616E+00 0.0016  9.91389E-01 0.0019  1.62324E+00 0.0013
  1.00290E+00 0.0018  1.82252E+00 0.0012  9.33297E-01 0.0018  1.91026E+00 0.0012
  8.21478E-01 0.0017  1.92116E+00 0.0011  7.46101E-01 0.0016  1.90231E+00 0.0011
  6.92607E-01 0.0015  1.85042E+00 0.0011  6.65601E-01 0.0015  1.79234E+00 0.0012
  6.67215E-01 0.0015  1.77085E+00 0.0012  6.69266E-01 0.0015  1.73432E+00 0.0012
  6.59504E-01 0.0015  1.66404E+00 0.0012  6.36304E-01 0.0015  1.56870E+00 0.0013
  6.03057E-01 0.0015  1.45011E+00 0.0013  5.78167E-01 0.0016  1.33789E+00 0.0014
  5.81856E-01 0.0017  1.27217E+00 0.0014  6.05060E-01 0.0018  1.21192E+00 0.0014
  6.45830E-01 0.0019  1.16380E+00 0.0015  7.11293E-01 0.0021  1.12309E+00 0.0016
  7.51070E-01 0.0021  1.08001E+00 0.0017  7.52337E-01 0.0022  1.02989E+00 0.0017
  7.34211E-01 0.0023  9.70775E-01 0.0018  7.07872E-01 0.0025  9.08306E-01 0.0019
  6.73626E-01 0.0027  8.46626E-01 0.0020  6.30408E-01 0.0032  7.82789E-01 0.0022
  5.89826E-01 0.0034  7.23625E-01 0.0024  5.48772E-01 0.0037  6.63037E-01 0.0025
  5.01637E-01 0.0039  5.97481E-01 0.0027  4.50037E-01 0.0043  5.38352E-01 0.0029
  3.99203E-01 0.0050  4.76715E-01 0.0032  3.48474E-01 0.0068  4.19819E-01 0.0036
  2.97868E-01 0.0084  3.62472E-01 0.0041  2.45023E-01 0.0102  3.06066E-01 0.0046
  1.86730E-01 0.0136  2.48456E-01 0.0052  1.22379E-01 0.0174  1.90881E-01 0.0056
  2.98067E-02 0.0475  1.36298E-01 0.0055  0.00000E+00 0.0000  9.78515E-02 0.0054
  0.00000E+00 0.0000  7.79275E-02 0.0057  0.00000E+00 0.0000  6.24088E-02 0.0061
  0.00000E+00 0.0000  5.15375E-02 0.0066  0.00000E+00 0.0000  4.35013E-02 0.0071
  0.00000E+00 0.0000  3.68060E-02 0.0076  0.00000E+00 0.0000  3.12940E-02 0.0083
  0.00000E+00 0.0000  2.69612E-02 0.0087  0.00000E+00 0.0000  2.33775E-02 0.0094
  0.00000E+00 0.0000  2.04072E-02 0.0103  0.00000E+00 0.0000  1.75141E-02 0.0109
  0.00000E+00 0.0000  1.50116E-02 0.0118  0.00000E+00 0.0000  1.30759E-02 0.0128
  0.00000E+00 0.0000  1.10920E-02 0.0138  0.00000E+00 0.0000  5.32575E-02 0.0552
  2.11129E+01 0.0008  3.92789E+01 0.0005
tfc   15       1       1       1      51       1       2       1       1
    2048000  3.93966E+01  1.72655E-03  1.09867E+03
    4096000  3.93128E+01  1.22054E-03  1.12550E+03
    6144000  3.93467E+01  1.05587E-03  1.01055E+03
    8192000  3.93316E+01  9.03429E-04  1.03925E+03
   10240000  3.93058E+01  8.01254E-04  1.05963E+03
   12288000  3.93160E+01  7.31162E-04  1.06229E+03
   14336000  3.92913E+01  6.75278E-04  1.06891E+03
   16384000  3.93006E+01  6.29361E-04  1.07765E+03
   18432000  3.93057E+01  5.92132E-04  1.08289E+03
   20480000  3.93018E+01  5.61377E-04  1.08493E+03
   22528000  3.92842E+01  5.33923E-04  1.09095E+03
   24576000  3.92879E+01  5.10401E-04  1.09469E+03
   26624000  3.92890E+01  4.89409E-04  1.09937E+03
   28672000  3.92785E+01  4.70783E-04  1.10310E+03
   30000000  3.92789E+01  4.59776E-04  1.10551E+03
```

Code 2. *Tally block.*

  – List of four numbers for each set of tally fluctuation data: number of hitories run when the sample was taken, tally, error and figure of merit

This is one of the simplest structures of a typical mctal file. In the example provided in Code 2, just one cell (namely cell 116) is requested for the presented tally. If more cells had been requested, a longer list would have been written to the mctal file. Instead, a completely different format for the cell list would have been produced if the user had requested a tally for a solid built with *macrobodies* or if the requested tally had been a *mesh* tally.

In the case of a tally with macrobody surfaces, the output format is shown in Code 3 and it is composed by two integers separated by a **.** (dot). The first number is the number of the macrobody defined in the input file and the second one is the number identifying the surface of the macrobody considered for the tally.

```
f        1
 1000.5
```

Code 3. *Cell list format when macrobodies are present.*

When a mesh tally is required, the information stored after the `f` card is again different, and it is shown in Code 4. In this case, the `f` keyword is once again followed by the number of cells, but it is also followed by three other numbers: the mesh tally type and the number of bins on the three available axes (set with the options `cora`, `corb`, `corc`). Then the list is composed by three blocks of floating-point numbers each containing the number of bins indicated in the previous line (in the example shown, three blocks with 100 numbers each).

2.2.1 Tally values nested structure

Particular attention should be devoted to the structure of the list of values of the tally, which is the the most subtle point in the reading procedure.

Consider again Code 2. In the tally presented there is just one cell, one direct bin, no user bins, 51 segments (50 + the total bin), no multipliers, two cosine bins and no energy and time bins. This means that for each cell, each segment and each cosine bin there is one value (with its relative error). Thus, in this example, 102 values are expected. MCNPX outputs these values according to Figure 1, i.e. the first written value corresponds to cell 1, segment 1, cosine bin 1, the second value corresponds to cell 1, segment 1, cosine bin 2, and so on.

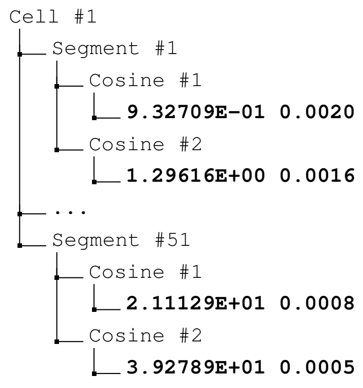

Figure 1. *Nested structure of tally values corresponding to Code 2.*

The nested structure of this example is now understood, but it can be further analyzed. In the case of energy and time bins, for example, the structure would be similar, but there would be different indexes (Figure 2).

It is now easy to understand that there are numerous possible arrangements of the multidimensional data structure and a general pattern should be found in order to be able to read each

```
f  1000000    0  100  100  100
 -6.50000E+01 -6.37000E+01 -6.24000E+01 -6.11000E+01 -5.98000E+01 -5.85000E+01
 -5.72000E+01 -5.59000E+01 -5.46000E+01 -5.33000E+01 -5.20000E+01 -5.07000E+01
 -4.94000E+01 -4.81000E+01 -4.68000E+01 -4.55000E+01 -4.42000E+01 -4.29000E+01
 -4.16000E+01 -4.03000E+01 -3.90000E+01 -3.77000E+01 -3.64000E+01 -3.51000E+01
 -3.38000E+01 -3.25000E+01 -3.12000E+01 -2.99000E+01 -2.86000E+01 -2.73000E+01
 -2.60000E+01 -2.47000E+01 -2.34000E+01 -2.21000E+01 -2.08000E+01 -1.95000E+01
 -1.82000E+01 -1.69000E+01 -1.56000E+01 -1.43000E+01 -1.30000E+01 -1.17000E+01
 -1.04000E+01 -9.10000E+00 -7.80000E+00 -6.50000E+00 -5.20000E+00 -3.90000E+00
 -2.60000E+00 -1.30000E+00 -5.90639E-14  1.30000E+00  2.60000E+00  3.90000E+00
  5.20000E+00  6.50000E+00  7.80000E+00  9.10000E+00  1.04000E+01  1.17000E+01
  1.30000E+01  1.43000E+01  1.56000E+01  1.69000E+01  1.82000E+01  1.95000E+01
  2.08000E+01  2.21000E+01  2.34000E+01  2.47000E+01  2.60000E+01  2.73000E+01
  2.86000E+01  2.99000E+01  3.12000E+01  3.25000E+01  3.38000E+01  3.51000E+01
  3.64000E+01  3.77000E+01  3.90000E+01  4.03000E+01  4.16000E+01  4.29000E+01
  4.42000E+01  4.55000E+01  4.68000E+01  4.81000E+01  4.94000E+01  5.07000E+01
  5.20000E+01  5.33000E+01  5.46000E+01  5.59000E+01  5.72000E+01  5.85000E+01
  5.98000E+01  6.11000E+01  6.24000E+01  6.37000E+01  6.50000E+01
 -7.50000E+01 -7.35000E+01 -7.20000E+01 -7.05000E+01 -6.90000E+01 -6.75000E+01
 -6.60000E+01 -6.45000E+01 -6.30000E+01 -6.15000E+01 -6.00000E+01 -5.85000E+01
 -5.70000E+01 -5.55000E+01 -5.40000E+01 -5.25000E+01 -5.10000E+01 -4.95000E+01
 -4.80000E+01 -4.65000E+01 -4.50000E+01 -4.35000E+01 -4.20000E+01 -4.05000E+01
 -3.90000E+01 -3.75000E+01 -3.60000E+01 -3.45000E+01 -3.30000E+01 -3.15000E+01
 -3.00000E+01 -2.85000E+01 -2.70000E+01 -2.55000E+01 -2.40000E+01 -2.25000E+01
 -2.10000E+01 -1.95000E+01 -1.80000E+01 -1.65000E+01 -1.50000E+01 -1.35000E+01
 -1.20000E+01 -1.05000E+01 -9.00000E+00 -7.50000E+00 -6.00000E+00 -4.50000E+00
 -3.00000E+00 -1.50000E+00  0.00000E+00  1.50000E+00  3.00000E+00  4.50000E+00
  6.00000E+00  7.50000E+00  9.00000E+00  1.05000E+01  1.20000E+01  1.35000E+01
  1.50000E+01  1.65000E+01  1.80000E+01  1.95000E+01  2.10000E+01  2.25000E+01
  2.40000E+01  2.55000E+01  2.70000E+01  2.85000E+01  3.00000E+01  3.15000E+01
  3.30000E+01  3.45000E+01  3.60000E+01  3.75000E+01  3.90000E+01  4.05000E+01
  4.20000E+01  4.35000E+01  4.50000E+01  4.65000E+01  4.80000E+01  4.95000E+01
  5.10000E+01  5.25000E+01  5.40000E+01  5.55000E+01  5.70000E+01  5.85000E+01
  6.00000E+01  6.15000E+01  6.30000E+01  6.45000E+01  6.60000E+01  6.75000E+01
  6.90000E+01  7.05000E+01  7.20000E+01  7.35000E+01  7.50000E+01
 -6.50000E+01 -6.37000E+01 -6.24000E+01 -6.11000E+01 -5.98000E+01 -5.85000E+01
 -5.72000E+01 -5.59000E+01 -5.46000E+01 -5.33000E+01 -5.20000E+01 -5.07000E+01
 -4.94000E+01 -4.81000E+01 -4.68000E+01 -4.55000E+01 -4.42000E+01 -4.29000E+01
 -4.16000E+01 -4.03000E+01 -3.90000E+01 -3.77000E+01 -3.64000E+01 -3.51000E+01
 -3.38000E+01 -3.25000E+01 -3.12000E+01 -2.99000E+01 -2.86000E+01 -2.73000E+01
 -2.60000E+01 -2.47000E+01 -2.34000E+01 -2.21000E+01 -2.08000E+01 -1.95000E+01
 -1.82000E+01 -1.69000E+01 -1.56000E+01 -1.43000E+01 -1.30000E+01 -1.17000E+01
 -1.04000E+01 -9.10000E+00 -7.80000E+00 -6.50000E+00 -5.20000E+00 -3.90000E+00
 -2.60000E+00 -1.30000E+00 -5.90639E-14  1.30000E+00  2.60000E+00  3.90000E+00
  5.20000E+00  6.50000E+00  7.80000E+00  9.10000E+00  1.04000E+01  1.17000E+01
  1.30000E+01  1.43000E+01  1.56000E+01  1.69000E+01  1.82000E+01  1.95000E+01
  2.08000E+01  2.21000E+01  2.34000E+01  2.47000E+01  2.60000E+01  2.73000E+01
  2.86000E+01  2.99000E+01  3.12000E+01  3.25000E+01  3.38000E+01  3.51000E+01
  3.64000E+01  3.77000E+01  3.90000E+01  4.03000E+01  4.16000E+01  4.29000E+01
  4.42000E+01  4.55000E+01  4.68000E+01  4.81000E+01  4.94000E+01  5.07000E+01
  5.20000E+01  5.33000E+01  5.46000E+01  5.59000E+01  5.72000E+01  5.85000E+01
  5.98000E+01  6.11000E+01  6.24000E+01  6.37000E+01  6.50000E+01
```

Code 4. *Cell list format when mesh tallies are present.*

```
• Cell #1
├── Energy #1
│   ├── Time #1
│   │   └── Value Error #1
│   ├── ...
│   └── Time #T
│       └── Value Error #N
├── ...
└── Energy #E
    ├── Time #1
    │   └── Value Error #E(T−1) + 1
    ├── ...
    └── Time #N
        └── Value Error #E·T
• ...
• Cell #F
├── ...
└── Energy #E
    └── Time #T
        └── Value Error #F·E·T
```

Figure 2. *Example of a nested structure of tally values.*

possible axes combination. This structure is presented in Figure 3 and in Section 3 its Python implementation will be discussed. The idea is to define a 12D matrix independently from the cards requested by the user and fill only the existing values.

```
Cell
└── Direct
    └── User
        └── Segment
            └── Multiplier
                └── Cosine
                    └── Energy
                        └── Time
                            └── i (for mesh cora card)
                                └── j (for mesh corb card)
                                    └── k (for mesh corc card)
                                        └── Value Error
```

Figure 3. *Complete tree for storing tally values.*

3 The mctal.py module

The Pyhton module `mctal.py` is the most important component of the converter, since it is the actual API to the mctal file. It contains the following classes:

- **Header**: this class is a container for storing the variables of the header (Section 2.1)
- **Tally**: this class is similar to the Header class and contains the variables related to the tally block (Section 2.2)

- **MCTAL**: this class contains an instance of the Header class and an array of Tally instances, one for each tally found in the mctal file. It also provides all the methods to read, parse and convert the mctal file into an object. It is basically the objective copy of the ASCII mctal file.

3.1 The Header class

In Code 5 the constructor of the Header class is presented with all the corresponding comments to help understanding the meaning of the variables.

```
def __init__(self,verbose=False):
    self.verbose = verbose                    # Verbosity flag
    self.kod     = ""                         # Name of the code, MCNPX
    self.ver     = ""                         # Code version
    self.probid  = np.array((), dtype=str)    # Date and time at problem runtime
    self.knod    = 0                          # The dump number
    self.nps     = 0                          # Number of histories that were run
    self.rnr     = 0                          # Number of pseudoradom numbers used
    self.title   = ""                         # Problem identification line
    self.ntal    = 0                          # Number of tallies
    self.ntals   = np.array((), dtype=int)    # Array of tally numbers
    self.npert   = 0                          # Number of perturbations
```

Code 5. *Constructor of the Header class.*

A function `Print()` is provided to quickly print to *stdout* all the class members.

3.2 The Tally class

The constructor of the Tally class, shown in Code 6 and Code 7, defines the variables required to store all the information contained in each tally. Arrays for storing axes bins are implemented only for cards allowing the use of user-defined bin values.

In Code 7, which still belongs to the constructor of the Tally class, some dictionaries and tuples are defined for storing verbose information to be used during the conversion.

A function `Print()` is provided also for class Tally and it uses the information in Code 7 to produce an easily readable dump of the Tally class members.

The most important part of the Tally class is shown in Code 8. The 12D data structure shown in Figure 3 is implemented using a NumPy array initialized with the `empty` method to reduce the initial memory consumption (NumPy arrays allow also to keep the memory footprint as small as possible even for huge mctal files, even if the time performance is slightly slower than standard Python lists).

The reason for defining a 12D data structure instead of a 9D one, as one would expect considering the 8 tally cards plus the value/error bin, is connected to the presence of mesh tallies. Since this MCNPX capability has been added after the definition of the 9D mctal file, and since mesh tallies define three new spatial coordinates (through `cora`, `corb` and `corc`), MCNPX developers adapted the structure of the mctal file to deal with this slightly different structure for stored data. It is not possible, in fact to define dir, usr, seg, mul, cos, erg and tim bins for mesh tallies, but they simply score the 3D spatial information and the value/error

```
def __init__(self,tN,verbose=False):
 self.verbose = verbose                         # Verbosity flag
 self.tallyNumber = tN                          # Tally number
 self.typeNumber = 0                            # Particle type number
 self.detectorType = None                       # The type of detector tally where
                                                # 0=none,1=point, 2=ring, 3=pinhole
                                                # radiograph, 4=transmitted image
                                                # radiograph (rectangular grid),
                                                # 5=transmitted image radiograph
                                                # (cylindrical grid)
                                                # When negative, it provides the
                                                # type of mesh tally

 self.radiograph = False                        # Flag set to True is the tally is
                                                # a radiograph tally.
 self.tallyParticles = np.array((), dtype=int)  # List of 0/1 entries indicating
                                                # which particles are used
 self.tallyComment   = np.array((), dtype=str)  # The FC card lines

 self.nCells = 0                                # Number of cell, surface or
                                                # detector bins

 self.mesh = False                              # True if the tally is a mesh tally
 self.meshInfo = np.array([0,1,1,1], dtype=int) # Mesh binning information in the
                                                # case of a mesh tally

 self.nDir = 0                                  # Number of total vs. direct or
                                                # flagged vs. unflagged bins

 self.nUsr = 0                                  # Number of user bins
 self.usrTC = None                              # Total / cumulative bin

 self.nSeg = 0                                  # Number of segment bins
 self.segTC = None                              # Total / cumulative bin

 self.nMul = 0                                  # Number of multiplier bins
 self.mulTC = None                              # Total / cumulative bin

 self.nCos = 0                                  # Number of cosine bins
 self.cosTC = None                              # Total / cumulative bin
 self.cosFlag = 0                               # The integer flag of cosine bins

 self.nErg = 0                                  # Number of energy bins
 self.ergTC = None                              # Total / cumulative bin
 self.ergFlag = 0                               # The integer flag of energy bins

 self.nTim = 0                                  # Number of time bins
 self.timTC = None                              # Total / cumulative bin
 self.timFlag = 0                               # The integer flag of time bins

 self.cells = np.array(())                      # Array of cell     bin boundaries
 self.usr = np.array(())                        # Array of user     bin boundaries
 self.seg = np.array(())                        # Array of segments bin boundaries
 self.cos = np.array(())                        # Array of cosine   bin boundaries
 self.erg = np.array(())                        # Array of energy   bin boundaries
 self.tim = np.array(())                        # Array of time     bin boundaries

 # For mesh tallies (or lattices)
 self.cora = np.array(())                       # Array of cora     bin boundaries
 self.corb = np.array(())                       # Array of corb     bin boundaries
 self.corc = np.array(())                       # Array of corc     bin boundaries

 self.tfc_jtf = np.array(())                    # List of numbers in the tfc line
 self.tfc_dat = []                              # Tally fluctuation chart data
                                                # (NPS, tally, error,
                                                # figure of merit)

 self.isInitialized = False
 self.valsErrors = None                         # Array of values and errors
```

Code 6. *Constructor of the Tally class. (Part I)* bin. This in principle could be dealt with just defining a smaller array, with four dimensions only. This solution, however, would require the duplication of the entire code of the API, since there would be two different possible data structures with different requirements. Instead, if a (8 + 3 + 1)D data structure is always initialized, it is possible to store only the relevant information, leaving the API code less error-prone, more readable and more efficient.

```
# Tuple with card names
self.binIndexList = ("f","d","u","s","m","c","e","t","i","j","k")

# Detector type names
self.detectorTypeList = { -6 : "smesh",                  # Short name
                          -5 : "cmesh",                  # Short name
                          -4 : "rmesh",                  # Short name
                          -3 : "Spherical mesh tally",
                          -2 : "Cylindrical mesh tally",
                          -1 : "Rectangular mesh tally",
                           0 : "None",
                           1 : "Point",
                           2 : "Ring",
                           3 : "Pinhole radiograph",
                           4 : "Transmitted image rdiograph (rectangular grid)" ,
                           5 : "Transmitted image radiograph (cylindrical grid)",
                           6 : "pi",                     # Short name
                           7 : "tir",                    # Short name
                           8 : "tic" }                   # Short name

# Short list of particles (used if self.typeNumber > 0)
self.particleListShort = { 1 : "Neutron",
                           2 : "Photon",
                           3 : "Neutron + Photon",
                           4 : "Electron",
                           5 : "Neutron + Electron",
                           6 : "Photon + Electron",
                           7 : "Neutron + Photon + Electron"}

# Complete list of particles (used to interpet the 0/1 list stored
# in self.tallyParticles)
self.particleList = ("Neutron", "Photon", "Electron", "Muon", "Tau",
                     "Electron Neutrino", "Muon Neutrino", "Tau Neutrino",
                     "Proton", "Lambda 0", "Sigma +", "Sigma -", "Cascade 0",
                     "Cascade -", "Omega -", "Lambda c +", "Cascade c +",
                     "Cascade c 0", "Lambda b 0", "Pion +", "Neutral Pion",
                     "Kaon +", "K0 Short", "K0 Long", "D +", "D 0", "D s +",
                     "B +", "B 0", "B s 0", "Deuteron", "Triton", "He3",
                     "He4 (Alpha)", "Heavy ions")
```

Code 7. *Constructor of the Tally class. (Part II)*

3.2.1 Methods and members of Tally class

As shown in Code 6, several tally properties are stored and can be easily accessed by the user. Some of them are simple variables and no interface functions are implemented, other contain a more complex set of information, thus requiring a pre-processing to provide the user with meaningful and consistent data.

Since Python does not divide class members between *public* and *private*, all Tally class members are accessible to the user. However, only some of them contain meaningful information and users are encouraged to take advantage of them. A list of these members is presented in Table 1.

In Table 2 interface methods are listed and commented.

3.3 The MCTAL class

The MCTAL class is the core of the `mctal.py` module. It uses the Header and Tally classes to read and store into objects the entire content of the mctal file. Its implementation contains

```
def initializeValuesVectors(self):

    nCells = self.getNbins("f")
    nCora  = self.getNbins("i")
    nCorb  = self.getNbins("j")
    nCorc  = self.getNbins("k")
    nDir   = self.getNbins("d")
    nUsr   = self.getNbins("u")
    nSeg   = self.getNbins("s")
    nMul   = self.getNbins("m")
    nCos   = self.getNbins("c")
    nErg   = self.getNbins("e")
    nTim   = self.getNbins("t")

    self.valsErrors = np.empty((nCells, nDir, nUsr, nSeg, nMul,
                                nCos, nErg, nTim, nCora, nCorb,
                                nCorc , 2 ), dtype=float)

    self.isInitialized = True
```

Code 8. *Constructor of the Header class.*

Variable name	Type	Content
`radiograph`	Boolean	`True` if radiograph tally, `False` otherwise
`mesh`	Boolean	`True` if mesh tally, `False` otherwise
`tallyNumber`	Integer	Tally number
`tallyComment`	String NumPy array	Tally comment line(s)
`binIndexList`	Tuple	Tally card letters

Table 1. *User-accessible Tally class members.*

Method name	Returned type	Accepted arguments
`getDetectorType()`	String	Boolean. If `True`, short names are returned (Code 7). By default the argument is set to `False`.
`getTallyParticles()`	String List	-
`getTotNumber()`	Integer	Boolean. If `True`, the total number of values is calculated considering also the total bin (if present). Default is `True`.
`getValue()`	Float	12 integers, one for each dimension: `f`, `d`, `u`, `s`, `m`, `c`, `e`, `t`, `i`, `j`, `k`, `v`. Note: `v` can only be 0 (value) or 1 (error).
`getAxis()`	Tuple	Char. The name of the axis can be chosen among: `u`, `s`, `c`, `e`, `t`, `i`, `j`, `k`
`getNbins()`	Integer	Char (axis name, as above) and Boolen for including or excluding the total bin (default is `True`

Table 2. *User-accessible Tally methods.*

essentially methods to parse text strings and it is designed to be as simple as possible to interface with. Also for this class, some members can be easily accessed by the user and they are listed in Table 3

Variable name	Type	Content
`thereAreNaNs`	Boolean	`True` if there is at least one `NaN` value in one of the tallies.
`header`	Header	Instance of the Header class (see Section 3.1).
`mctalFileName`	String	Name of the analyzed mctal file.

Table 3. *User-accessible MCTAL class members.*

Table 4 shows the only function provided to users. The function `Read()` is in charge of reading, parsing and storing into objects the content of the mctal file and returns a list of Tally class instances. No information about the header is returned by this method.

Method name	Returned type	Accepted arguments
`Read()`	List of Tally instances	-

Table 4. *User-accessible MCTAL methods.*

When the `Read()` method is called by the user, an instance of the Header class is created and filled with corresponding data. Then, the method enters a loop that terminates when EOF is reached (or if some error occurs). Each step in the loop creates a new instance of the Tally class, parses the corresponding tally, then stores the class in a list, which is returned after the loop is complete.

4 Interfacing with the API

Interfacing with the mctal API does not require complex coding. It is as simple as importing the module, making an instance of the MCTAL class and call the `Read()` method to get a list of all the tallies in the mctal file.

```
from mctal import MCTAL

m = MCTAL(mctalFile,verbose)
T = m.Read()
```

Code 9. *Interfacing with the API*

Note in Code 9 that the MCTAL constructor takes two arguments: the first one is the name of the mctal file and the second one is a boolean flag to enable or disable messages to standard output during the import process. By default this parameter is set to `False`.

One possible application of this API is now presented and it is intended to convert mctal files to the ROOT binary format.

4.1 The ROOT converter

The first main application of the MCTAL API is to build a mctal to ROOT converter, in order to take full advantage of the data analysis framework developed at CERN, which is becoming increasingly popular. Without digging into the detail of the converter, the concept is presented just to show how it is possible to take advantage of the various members and methods provided by the API.

Once tat the MCTAL instance is created and the tallies are read it is possible to access each of them using a for loop and in Codes 10-12 a possible routine for process them is presented.

```
for tally in T:

    tallyLetter = "f"

    if tally.radiograph:
          tallyLetter = tally.getDetectorType(True)

    if tally.mesh:
          tallyLetter = tally.getDetectorType(True)

    nCells = tally.getNbins("f",False)

    ...

    ergAxis = tally.getAxis("e")
    nErg = tally.getNbins("e",False)

    timAxis = tally.getAxis("t")
    nTim = tally.getNbins("t",False)
```

Code 10. *Concept of converter to ROOT (part I)*

In Code 10 the method `getDetectorType()` is used to retrieve the short name of the current tally, which will be used as the tally name in the converted ROOT file. Tally names are composed using one or more letters and the corresponding number. The default letter is `f` and, if the tally is a mesh or a radiograph, then the text part of the name is set accordingly.

After the tally name, the number of bin is retrieved using the `getNbins()` method, to which the letter of the axis is passed. The second argument, set to `False`, prevents from considering the total bin (if present), since the total value will not be exported. For binnings which also have the possibility to use specific values, like energy or time, the `getAxis()` method is used to obtain a NumPy array with bin values, that will be set as axes values once converted.

```
if tally.mesh == True:

     bins    = np.array( [nCells, nDir, nUsr, nSeg, nMul, nCos, nErg, nTim, nCora,
                          nCorb, nCorc], dtype=np.dtype('i4') )
     binsMin = np.array( [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.] )
     binsMax = np.array( [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.] )

     hs = ROOT.THnSparseF("%s%d" % (tallyLetter, tally.tallyNumber),
                          string.join(tally.tallyComment.tolist()).strip(), 11,
                          bins, binsMin, binsMax)

else:

     bins    = np.array( [nCells, nDir, nUsr, nSeg, nMul, nCos, nErg, nTim],
                          dtype=np.dtype('i4') )
     binsMin = np.array( [0., 0., 0., 0., 0., 0., 0., 0.] )
     binsMax = np.array( [1., 1., 1., 1., 1., 1., 1., 1.] )

     hs = ROOT.THnSparseF("%s%d" % (tallyLetter, tally.tallyNumber),
                          string.join(tally.tallyComment.tolist()).strip(), 8,
                          bins, binsMin, binsMax)

...

if len(ergAxis) != 0:
     hs.GetAxis(6).Set(nErg,ergAxis)
```

Code 11. *Concept of converter to ROOT (Part II)*

In Code 11 the ROOT sparse histogram (THnSparseF) is initialized accordingly to the tally type: if the tally is a mesh, the ROOT histogram will have 11 dimensions, otherwise just 8. Moreover, the tally comment line(s) stored in the `tallyComment` list are used as the histogram title. Then, for each axis that can have specific values, the NumPy array obtained with the `getAxis()` method is set as the histogram axis.

```
for f in range(nCells):
   for d in range(nDir):
      for u in range(nUsr):
         for s in range(nSeg):
            for m in range(nMul):
               for c in range(nCos):
                  for e in range(nErg):
                     for t in range(nTim):
                        for k in range(nCorc):
                           for j in range(nCorb):
                              for i in range(nCora):

                                 val = tally.getValue(f,d,u,s,m,c,e,t,i,j,k,0)
                                 err = tally.getValue(f,d,u,s,m,c,e,t,i,j,k,1)

                                 if tally.mesh == True:

                                    hs.SetBinContent(np.array([f+1,d+1,u+1,s+1,
                                                     m+1,c+1,e+1,t+1,i+1,j+1,k+1],
                                                     dtype=np.dtype('i4')),val)

                                    hs.SetBinError(np.array([f+1,d+1,u+1,s+1,
                                                     m+1,c+1,e+1,t+1,i+1,j+1,k+1],
                                                     dtype=np.dtype('i4')),val*err)

                                 else:

                                    hs.SetBinContent(np.array([f+1,d+1,u+1,s+1,
                                                     m+1,c+1,e+1,t+1],
                                                     dtype=np.dtype('i4')), val)

                                    hs.SetBinError(np.array([f+1,d+1,u+1,s+1,
                                                     m+1,c+1,e+1,t+1,i+1,j+1,k+1],
                                                     dtype=np.dtype('i4')),val*err)
```

Code 12. *Concept of converter to ROOT (Part III)*

The last part, shown in Code 12, is the filling of the histogram and this is done through a series of nested for cycles. Errors, which in the mctal file are relative errors, are converted into absolute errors and set to each histogram bin.

5 Supported and unsupported features

Since the structure of a mctal file is complex and may contain a lot of information, the first version of this API could not address all of the possible features of mctal files. However, this first implementation is able to deal with most of them.

A list of supported features is now presented:

- all possible binnings
- macrobodies
- mesh tallies
- radiograph tallies
- tfc data

The following list contains the features unsupported by the API (i.e. they are not read and not imported in the MCTAL object):

- `pert` card
- `KCODE` card
- the presence of at least one `NaN` value, sets a flag to the entire MCTAL instance, in order to skip tests

Acknowledgements

The author would aknowledge the entire Neutronics Group at ESS, in particular Dr. Batkov and Dr. Zanini, for making this work possible. Particular thanks are also reserved to Dr. Bevilacqua, Esben Klinkby and Troels Shönfeldt for being the first enthusiastic users and for providing important suggestions and a huge amount of test files.
This work was entirely funded by the European Spallation Souce.

References

[1] D. Pelowitz (Ed.), *MCNPX User's Manual, Version 2.5.0*. Los Alamos National Laboratory LA-CP-05-0369, April 2005.

[2] D. Pelowitz (Ed.), *MCNPX User's Manual, Version 2.6.0*. Los Alamos National Laboratory LA-CP-07-1473, April 2009.

[3] D. Pelowitz (Ed.), *MCNPX User's Manual, Version 2.7.0*. Los Alamos National Laboratory LA-CP-11-00438, April 2011.

[4] LANL, RSICC, *CCC-740 MCNPX 2.7.0 (C00740MNYCP08)*. 2011.

[5] *F. Gallmeier private communications*. 2010.