EPJ Web of Conferences **146**, 02044 (2017) DOI: 10.1051/epjconf/201714602044
ND2016

Development of a new nuclear data library based on ROOT

Tae-Sun Park[1,2,a], Kyung Joo Min[3], Eun Jin In[3], Sang-In Bak[3], and Seung-Woo Hong[3]

[1] Department of Physics, Sungkyunkwan University, Suwon, Korea
[2] Institute of Basic Science, Sungkyunkwan University, Suwon, Korea
[3] Department of Energy Science, Sungkyunkwan University, Suwon, Korea

Abstract. We develop a new C++ nuclear data library for the Evaluated Nuclear Data File (ENDF) data, which we refer to as TNudy. Main motivation of the development is to provide systematic, powerful and intuitive interfaces and functionalities for browsing, visualizing and manipulating the detailed information embodied in the ENDF. To achieve this aim efficiently, the TNudy project is based on the ROOT system. TNudy is still in the stage of development, and the current status and future plans will be presented.

1. Introduction

Evaluated Nuclear Data File or ENDF [1,2] is a format for evaluated nuclear reaction data on cross sections, angular and energy distributions of secondary particles, resonance parameters, the numbers of prompt and delayed neutrons per fission, fission product yields, the covariance data, and so on. There are several tools and programs implementing ENDF. For example, JANIS [3] is equipped with an intuitive graphic user interface (GUI) with many useful functionalities of navigating and presenting the data. PREPRO [4] and NJOY [5] can convert the ENDF/B data into the form suitable for applications. However, there are not many computer software libraries for ENDF data. Here, what we mean by "library" is a collection of subroutines, classes and other resources used by computer programs, which should not be confused with an ENDF data set. A library has a unique feature that it can be plugged into users' own codes. Thus, a library equipped with standardized accessing methods and functionalities relevant to the data structures of ENDF will be extremely powerful and flexible in achieving users' diverse needs in a variety of situations. Such a library can also be implemented in developing other programs for ENDF.

With these motivations, we began to develop a C++ library for ENDF, aiming at the construction of systematic, versatile, powerful and intuitive interfaces and functionalities for browsing, visualizing and manipulating the detailed information embodied in the ENDF. To achieve this aim efficiently, this project uses a powerful system called the ROOT [6], which has been developed to treat and analyze a vast amount of data in nuclear and particle physics experiments. The advantage of this way is that we can make use of the vast amount of well-tested well-organized functionalities of ROOT. Our project is named as "TNudy", which stands for "rooT NUclear Data librarY" [7]. Below we will describe the structures and features of TNudy, and report on the current status of the development.

[a] e-mail: tspark@kias.re.kr

2. Structure of TNudy

2.1. Building blocks

The most elementary class of TNudy is TNuObject class, which is inherited from the TObject class of the ROOT. Its role is to provide the interfaces and functionalities common to classes of TNudy.

ENDF data are presented by six types of records, which are CONT, LIST, TAB1, TAB2, TEXT and INTG. The first four record types are implemented as TNuCont, TNuList, TNuTab1 and TNuEndfTAB2, respectively. Collections of TEXT and INTG are implemented as TNuDescription and TNuIntg, respectively.

ENDF data are basically collections of the above records, and the base class for such a collection is TNuRecs. It contains the member functions such as GetSize(), GetEntries() and At(int i). It can contain any objects that are derived from the TNuObject, that is, its entry can be an instance of TNuRecs for a sub-structure made of records as well as an instance for a single record. This property enables us to have a hierarchical structure in quite an intuitive manner. Corresponding to various structures of ENDF data, there are many classes that are derived from TNuRecs. For example, TNuRecsWithCont and TNuRecsWithTab1 classes are for the collection of records with a header record of type CONT and TAB1, respectively. The header record contains information such as the number of entries and parameters of the structure.

Apart from these classes which correspond to distinct physical structures of data, we have also implemented several *abstract* classes following the functionalities of the data. For example, TNuVTab class is for the interpolation of the tabulated data, TNuVF1 and TNuVF2 classes are for 1- and 2-dimensional functors, and so on. By combing these classes with the use of the inheritance technique of C++, we can implement a large number classes in a cost-effective manner both in development and in management, avoiding unnecessary and undesirable duplications in code. Here let us show one example, the TNuTab2 class, which is for the two-dimensional functor

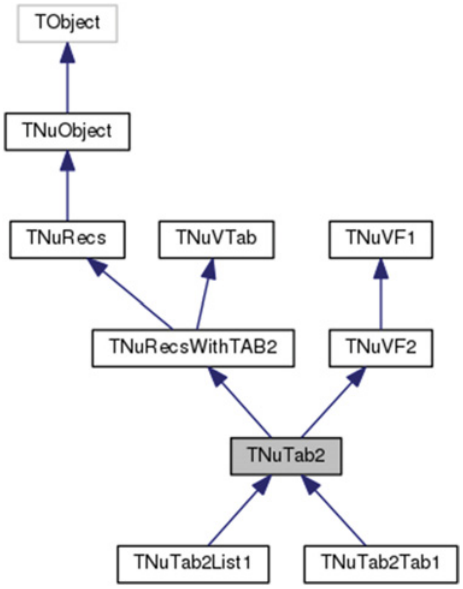


Figure 1. The inheritance diagram of `TNuTab2`.

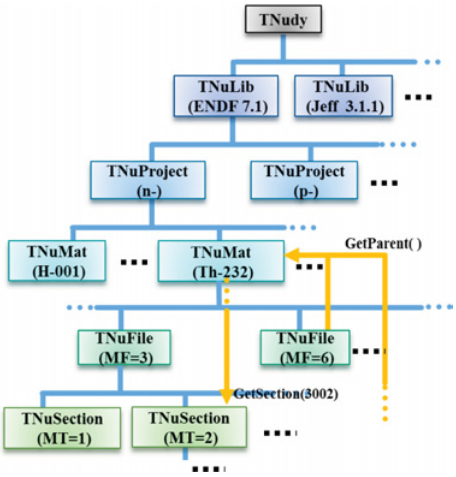


Figure 2. The hierarchy structure and accessing methods of TNudy.

with the ENDF TAB2 record. The inheritance diagram of it is drawn in Fig. 1.

2.2. Hierarchy

As is shown in Fig. 2, TNudy has the same hierarchical structure as the ENDF: TNuLib class is for a collection of data from an evaluation group, TNuSublib class is for the sub-library data of a incident particle, TNuMat class is for the data of a target material, TNuFile class is for a block of data of a certain data type, and TNuSection class is for the data of a reaction channel. Since the structure of TNuSection depends on the "file number" MF and sometimes also on the "section number" MT, and we have dozens of classes derived from TNuSection class. For example, TNuSec03 class is for MF=3 (total cross sections), TNuSec08454 class is for MF = 8 and MT = 454 (fission product yield data), TNuSec08457 class is for MF = 8 and MT = 457 (spontaneous fission data), and so on. Each of these derived TNuSection class has its own subsection structure.

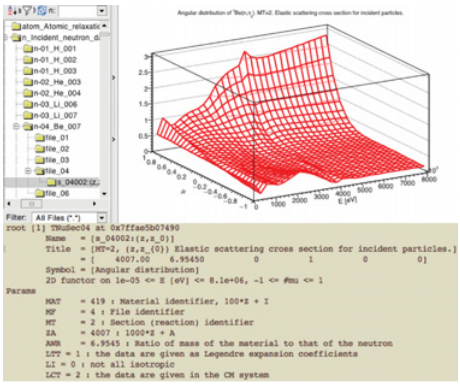


Figure 3. An example of TBrowser usage with TNudy.

3. Access of objects

Since ENDF is a collection of data with a hierarchy, it is very important to have efficient methods of accessing objects.

The most intuitive and easiest way is to use the graphic user-interface of ROOT, TBrowser. As is shown in Fig. 3, one can navigate easily ENDF data with a mouse click. And many of the relevant methods of visualizing the contents such as drawing 1D or 2D graphs can also be accessed from the browser panel. Furthermore, the command line interface of the ROOT which can interpret most of C++ grammars, cint, is a very convenient tool to run TNudy. TNudy has several handy methods that can be used in the cint environment such as `ls()` and `What()`, which print the table of contents and a brief explanation of the structure, respectively. Figure 3 shows the results of `DrawDXDmuNormalized` (top) and `What` functions (bottom), saying that the graph is the angular distribution of the elastic $n + {}^{9}\mathrm{Be}$ scattering measured in the CM system, for the 2D region in the ranges of $E_n = (10^{-5} \sim 8.1 \times 10^{6})$ eV and $\cos\theta = (-1 \sim 1)$.

For the access of data in a macro or a code, TNudy has many intuitive methods. For example, `TNuLib::GetSublib("n")` returns the sub-library for the neutron-incident data, `TNuSublib::GetMat("th232")` or `TNuSublib::GetMat(90, 232)` returns the TNuMat instance for ^{232}Th, `TNuMat::GetSection(3, 2)` returns the TNuSection instance with MF=3 and MT=2 (which corresponds to the elastic cross section), as represented in Fig. 2. But accessing sub-section data is problematic, as they have diverse kinds of structures with different depths of hierarchy levels. To resolve the situation, we have implemented several features and methods in TNudy.

Most importantly, we make all the classes for the section and sub-section structure inherit from the aforementioned `TNuRecs` class, which provides us with standard methods to iterate and to access the enclosed entities. Thus, one can use the following code to iterate the whole entities of an instance of `TNuRecs`:

```
for (int i = 0; i < recs->GetSize(); i++) {
    TNuObject* obj = recs->At(i);
    if (obj) { // obj can be null
      // user specific codes ...
     }
}
```

where `GetSize()` and `At(i)` return the size of the collection and the i-th entity of the instance.

To provide a more abstract way of iterating a collection, TNudy contains also `GetCollection()` method which enables us to create the standard iterator of ROOT, `TIter`. Then the above code can be written as follows:

```
TObject* obj;
TIter next(recs->GetCollection());
while ((obj=(TNuObject*) next())) {
    // user specific codes ...
}
```

It is often necessary to find the parent of a given object, the container to which the object belongs to. For such a case, TNudy has `GetParent()` method which returns the pointer of the parent. This feature enables us to trace back all the parents of an object located in any level in the hierarchy.

It should be emphasized that the above methods are common to all the sub-structure of the ENDF, independent of the detailed structure and the hierarchical depth of the data.

4. Other important features

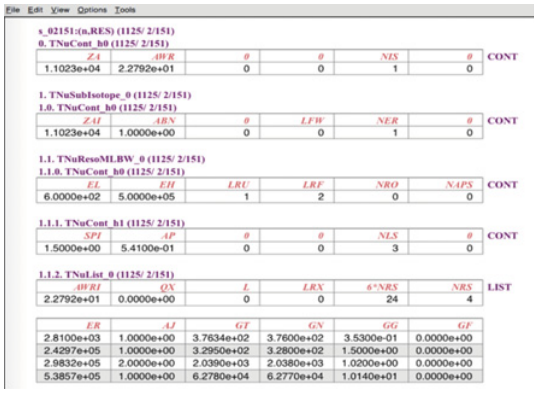
Load-on-demand or lazy-loading is implemented in TNudy through `TNuLazyCollection` class. An instance of this class does not load its member data into memory when it is created. Instead, it loads only the necessary amount of data on-demand. For example, when `At(i)` is invoked, it then loads the data for the i-th entity, without touching other parts of the data. The TNuLib, TNuSublib, TNuMat and TNuFile inherit from this class. Thanks to this feature, for example, one can promptly search and access a part of data of huge-sized TENDL-2015, without wasting time and memory for loading the whole data.

In understanding the meaning of a ENDF record, having the labels of each field of the record is very helpful. TNudy stores all the labels. Furthermore, for each record and sub-structured object, TNudy assigns and stores a name (which is required by TBrowser) and a title, a string that briefly explains the object. By noting that these data are highly duplicative, we could in fact store them in a very cost-effective way. Combined with `PrintOnCanvas()` which prints the ENDF data on a graphic canvas, these extra information enables us to write the ENDF data in an human-readable way as can be seen in Fig. 4.

5. Discussions

As of now TNudy consists of about 130 classes, and most of the aimed functionalities have been implemented. That is, it is capable of building cross sections based on the resonance information given in File 2. And it can also draw 1D and 2D graphs of 2D or 3D data by fixing one or two arguments to specific values.

Here we acknowledge that TNudy has been successfully used in the development of a new data-based hadronic


File Edit View Options Tools

s_02151:(n,RES) (1125/ 2/151)
0. TNuCont_h0 (1125/ 2/151)

ZA	AWR	0	0	NIS	0	
1.1023e+04	2.2792e+01	0	0	1	0	CONT

1. TNuSubIsotope_0 (1125/ 2/151)
1.0. TNuCont_h0 (1125/ 2/151)

ZAI	ARN	0	LFW	NER	0	
1.1023e+04	1.0000e+00	0	0	1	0	CONT

1.1. TNuResoMLBW_0 (1125/ 2/151)
1.1.0. TNuCont_h0 (1125/ 2/151)

EL	EH	LRU	LRF	NRO	NAPS	
6.0000e+02	5.0000e+05	1	2	0	0	CONT

1.1.1. TNuCont_h1 (1125/ 2/151)

SPI	AP	0	0	NLS	0	
1.5000e+00	5.4100e-01	0	0	3	0	CONT

1.1.2. TNuList_0 (1125/ 2/151)

AWRI	QX	L	LRX	6*NRS	NRS	
2.2792e+01	0.0000e+00	0	0	24	4	LIST

ER	AJ	GT	GN	GG	GF
2.8100e+03	1.0000e+00	3.7634e+02	3.7600e+02	3.5300e-01	0.0000e+00
2.4297e+05	1.0000e+00	3.2950e+02	3.2800e+02	1.5000e+00	0.0000e+00
2.9832e+05	2.0000e+00	2.0390e+03	2.0380e+03	1.0200e+00	0.0000e+00
5.3857e+05	1.0000e+00	6.2780e+04	6.2770e+04	1.0140e+01	0.0000e+00

Figure 4. An example of `PrintOnCanvas`, which shows the ability of TNudy of printing the name, title and labels of the data.

model of Geant4 dedicated for accurate description of the neutron production through the ^{9}Be$(p, n)^{9}$B reaction [8], the goal of which has been achieved by taking the ENDF./B-VII.1 data of the reaction as inputs of the model.

We are currently in the debugging phase, and we hope that it will be mature enough to be open to public in a near future.

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2013R1A1A2063824, NRF-2015M2A2A4A01045320, NRF-2015M2B2A9032869).

References

[1] `http://www.nndc.bnl.gov/exfor/endf00.jsp`, `https://www-nds.iaea.org/exfor/endf.htm`
[2] `https://www.oecd-nea.org/dbdata/data/nds_eval_libs.htm`
[3] A. Nouri et al., "JANIS: A New Software for Nuclear Data Services," Proc. Int. Conf. On Nuclear Data for Science and Technology, Tsukuba, Japan, October 2001, p. 1480 (2002)
[4] D.E. Cullen, PREPRO 2007–2007 ENDF/B Pre-processing Codes, report IAEA-NDS-39, Rev. 13, March 2007, "`https://www-nds.iaea.org/public/endf/prepro/`"
[5] R.E. MacFarlane, D.W. Muir, The NJOY Nuclear Data Processing System, Version 91, LA-12740-M (1994)
[6] R. Brun and F. Rademakers, ROOT – An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. **A389**, 81 (1997). See also `http://root.cern.ch/`
[7] S. I. Bak, R. Brun, F. Carminati, J. S. Chai, A. Gheata, M. Gheata, S.-W. Hong, Y. Kadi, V. Manchanda, T.-S. Park, C. Tenreiro, A New Format for Handling Nuclear Data, J. Korean Phys. Soc. **59**, 1111 (2011)
[8] J.-W. Shin and T.-S. Park, Nucl. Inst. & Meth. **B342**, 194 (2015)