



LAWRENCE  
LIVERMORE  
NATIONAL  
LABORATORY

# Generalized Nuclear Data: a New Structure (with Supporting Infrastructure) for Handling Nuclear Data

C. M. Mattoon, B. R. Beck, N. R. Patel, N. C. Summers, G. W. Hedstrom, D. A. Brown

September 25, 2012

Nuclear Data Sheets

## **Disclaimer**

---

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

# Generalized Nuclear Data: a New Structure (with Supporting Infrastructure) for Handling Nuclear Data

C.M. Mattoon\*, B.R. Beck, N.R. Patel, N.C. Summers, G.W. Hedstrom  
*Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore CA, United States*

D.A. Brown  
*National Nuclear Data Center, Upton NY, United States*  
(Dated: September 13, 2012)

The Evaluated Nuclear Data File (ENDF) format was designed in the 1960s to accommodate neutron reaction data to support nuclear engineering applications in power, national security and criticality safety. Over the years, the scope of the format has been extended to handle many other kinds of data including charged particle, decay, atomic, photo-nuclear and thermal neutron scattering. Although ENDF has wide acceptance and support for many data types, its limited support for correlated particle emission, limited numeric precision, and general lack of extensibility mean that the nuclear data community cannot take advantage of many emerging opportunities. More generally, the ENDF format provides an unfriendly environment that makes it difficult for new data evaluators and users to create and access nuclear data.

The Cross Section Evaluation Working Group (CSEWG) has begun the design of a new Generalized Nuclear Data (or ‘GND’) structure, meant to replace older formats with a hierarchy that mirrors the underlying physics, and is aligned with modern coding and database practices.

In support of this new structure, Lawrence Livermore National Laboratory (LLNL) has updated its nuclear data/reactions management package FUDGE to handle GND structured nuclear data. FUDGE provides tools for converting both the latest ENDF format (ENDF-6) and the LLNL Evaluated Nuclear Data Library (ENDL) format to and from GND, as well as for visualizing, modifying and processing (i.e., converting evaluated nuclear data into a form more suitable to transport codes) GND structured nuclear data.

GND defines the structure needed for storing nuclear data evaluations and the type of data that needs to be stored. But unlike ENDF and ENDL, GND does not define how the data are to be stored in a file. Currently, FUDGE writes the structured GND data to a file using the eXtensible Markup Language (XML), as it is ASCII based and can be viewed with any text editor. XML is a meta-language, meaning that it has a primitive set of definitions for representing hierarchical data/text in a file. Other meta-languages, like HDF5 which stores the data in binary form, can also be used to store GND in a file.

In this paper, we will present an overview of the new GND data structures along with associated tools in FUDGE.

	<b>Contents</b>		
		A. Forms	6
		B. Components	7
<b>I. Introduction</b>	2	C. Links	8
<b>II. Overview of GND</b>	4	<b>V. Details of the GND structure</b>	8
<b>III. Naming convention</b>	5	A. The ‘style’ element	8
A. Particle and product naming	5	B. The ‘documentations’ element	8
B. Reactions	5	C. The ‘particles’ element	8
<b>IV. Representations of data in GND</b>	6	D. ‘resonances’ element	9
		E. ‘reaction’ element	9
		1. The ‘crossSection’ element	10
		2. The ‘outputChannel’	10
		F. ‘product’ elements	11
		1. ‘distributions’	11
		2. ‘multiplicity’	11

\*Corresponding author, electronic address: mattoon1@llnl.gov

3. Other product data	12
G. Other top level elements	12
H. Covariances	13
<b>VI. General-purpose data containers for GND</b>	14
A. Axes element	14
B. One-dimensional function containers	14
C. Two-dimensional function containers	15
D. Three-dimensional function containers	15
E. Legendre containers	16
F. Table and matrix containers	16
<b>VII. Associated Tools for GND</b>	16
A. Resonance Reconstruction	17
B. ‘toPointwiseLinear’	17
C. Checking codes	17
1. XML Schema	18
2. Checking physics content	18
D. Plotting Tools	19
<b>VIII. Translating ENDF-6 formatted files to and from GND</b>	20
A. Converting pointwise to piecewise	20
B. Particle masses	21
C. Handling ENDF-6 format errors	22
<b>IX. Future Work</b>	22
A. Finalizing the Structure of GND	22
B. Processing codes for GND	23
C. Transport code access routines (GIDI)	23
D. Uncertainty quantification tools: Monte Carlo and deterministic uncertainty propagation	24
E. Evaluator tools: creating new GND evaluations directly from Nuclear Reaction Modeling Codes	24
F. External particle database	24
<b>X. Conclusion</b>	25
<b>Acknowledgments</b>	26
<b>References</b>	26

## I. INTRODUCTION

The field of nuclear data lies at the junction between basic nuclear physics, computational physics, and engineering. The scientists and engineers who generate nuclear data (the ‘nuclear data community’) are responsible for producing reliable, up-to-date information about nuclear reactions, structures and decays in a computer-readable form that can be integrated in computer simulations of nuclear reactors, medical physics applications, nuclear weapons, etc. Data needed for these applications include reaction cross sections, energy and angle distributions and multiplicities of outgoing products, as well

as uncertainties and covariance matrices. Additionally, these applications often call for detailed nuclear structure information (e.g. for identifying materials based on gamma ray emissions) and decay information (for dosimetry and decay heat calculations).

The Evaluated Nuclear Data File (ENDF) [1] format was designed in the 1960s to accommodate neutron reaction data to support nuclear engineering applications in power, national security and criticality safety. Over the years, the scope of the format has been extended to handle charged particle, photo-nuclear, decay, atomic and thermal neutron data. Along the way, many inconsistencies and oddities have crept into the format as the nuclear data community has sought to shoe-horn new data into the ENDF format. The ENDF format is currently on version 6 (ENDF-6), and there are many tools for processing, creating, testing and visualizing ENDF-formatted evaluations. Since ENDF has been adopted by most of the international nuclear data community, it is the format of choice for data centers to exchange evaluated data.

Despite its widespread acceptance, the ENDF format is showing signs of its age. New applications in medical physics, space/radiation physics, homeland security, nuclear waste management and non-proliferation drive ENDF far beyond its original design and core uses. The limited support in ENDF for correlated particle emission, limited numeric precision, and general lack of extensibility mean that the nuclear data community must currently make approximations to the basic nuclear data that reduce their ability to address many new opportunities. More critically, at a time when the nuclear data community is facing manpower shortages, new entrants to the field find that learning the ENDF format is a steep barrier that must be overcome in order to gain expertise in the field. Together, these new needs and users of nuclear data drive the need to simplify and modernize the ENDF format.

The ENDF format is not necessarily *broken*, but modernizing it is a daunting and expensive task that should not be undertaken lightly. Here the nuclear data community is helped by one of its own, Lawrence Livermore National Laboratory (LLNL). LLNL was forced to revamp its own internal data format used for its Evaluated Nuclear Data Library (ENDL). The ENDL format pre-dates the ENDF format and suffers from many of the same shortcomings. To address these, LLNL used funding from the US Department of Energy’s Advanced Strategic Computing (ASC) program to write a new infrastructure for nuclear data management called FUDGE. This ASC funding and the American Recovery and Reinvestment Act allowed LLNL to build on this core effort and create the Generalized Nuclear Data structure (GND) and supporting tools, including the ability to convert from ENDF-6 to GND and vice versa.

The Generalized Nuclear Data project allows the nuclear data community to revisit the design of our common nuclear data format, “righting many shortcomings” in the format, before critical expertise in the field disappears

due to retirements. Fortunately, the past 50 years have seen tremendous growth in computer technology, leading not only to more powerful hardware but also to new programming languages and concepts. Leveraging this work, we propose storing nuclear data in a simple and easily understood hierarchy that can be manipulated and used with off-the-shelf open source software. The main goals that influence the design of GND are to:

- **Define a *structure* for storing nuclear data, not a particular *implementation*.** In other words, while ENDF-6 defines both a data structure and how that structure is to be stored in a file, GND defines the outline for how data are to be represented, but does not dictate the type of data storage. The data storage type could be any ‘meta-language’ such as XML or HDF5 (a meta-language defines constructs for building a language or format).
- **Design the structure to mirror the underlying nuclear reaction physics, making the data easier to understand and use.**
- **Continue supporting all parameterized forms available in ENDF**, including resonance parameters, Maxwellian and Watt spectra, and many others. This allows for efficient storage of data, and also leverages the man-decades of experience invested in the processing and handling of parameterized types in ENDF.
- **Define a small set of general-purpose data containers that can be used to store various data types.** These are similar to the low-level data containers that appear in ENDF (such as the List, TAB1 and TAB2 containers described in Section 0.6.3 of reference [1]). These new multi-purpose data containers are designed to be easily adapted for use in other data projects, including possible future formats for storing Nuclear Structure information and experimental Nuclear Data.
- **Reduce or eliminate artificial limitations.** For example, due to the limited space for reaction identifiers, the ENDF format can only handle 40 discrete neutron inelastic scattering reactions (MTs 51 to 90), whereas GND allows for an arbitrary number and type of reactions.
- **Reduce redundant data and the possibility for discrepancies, by storing data in only one place and referring to it using a name or link.**
- **Support storing evaluated data and derived data simultaneously.** For example, cross section data may be stored as resonance parameters with background, and also reconstructed into pointwise or grouped data. In GND, these forms may be stored together in the same structure, simplifying

data management. This also allows for a shared infrastructure, including a common interface for accessing evaluated and derived data.

The final two goals in this list may appear contradictory at first glance: how can GND simultaneously store multiple forms of data, but not store redundant data? The distinction is that, while the original evaluated data should always avoid redundancy, one or more forms of derived data (produced by processing the original data) can be stored alongside the original. GND clearly indicates which data are the original using the ‘nativeData’ attribute described in Sec. IV.

In order to meet all these goals, a flexible underlying design is required. This has led to conceiving GND as a nested hierarchy, similar to a computer file system. This hierarchy has three essential building blocks: ‘dataset’, ‘element’ and ‘attribute’. A ‘dataset’ is similar to a file in a file system. An ‘element’ is similar to a directory in a file system in that it can contain datasets as well as other elements. Elements may also have ‘attributes’ associated with them. Attributes are small chunks of data that describe the element to which they are attached (attributes are also sometimes called ‘metadata’, and are supported by some file systems). These three ingredients, ‘dataset’, ‘element’ and ‘attribute’, are all that are required of a meta-language in order to implement the GND structure. Table I shows how these building blocks map to other meta-language building blocks.

TABLE I: Names of common building-blocks that appear in GND, compared with their names in some common meta-languages.

GND	File System	XML	HDF5
dataset	file	text	dataset
element	directory	element	group
attribute	meta-data	attribute	attribute

The reader will note that GND is called a structure rather than a format. When GND is represented in a meta-language like XML or HDF5, the resulting file is said to be formatted in GND/XML or GND/HDF5 respectively. Because XML and HDF5 both support the essential building blocks needed for GND, the conversion from a GND/XML file to a GND/HDF5 file and vice versa is simple.

In this paper, we will be referring to some specific implementations of GND, in the Python programming language and also in the meta-languages XML [2] and HDF5 [3], all of which provide the necessary ingredients for storing GND data. We wish to re-emphasize, however, that these are only particular implementations and that the GND structure can be expressed in other meta-languages as well.

A nuclear data/reaction management package called FUDGE [4], developed by Lawrence Livermore National Laboratory, has been updated to support the GND structure. FUDGE (For Updating Data and Generating Eval-

uations) is primarily written in Python, and can read, write, visualize, modify, check and process data stored in GND. The code is available for free download from the Nuclear Data collaboration server at <https://ndc1x4.bnl.gov/gf/project/gnd>.

FUDGE has recently been extended with the ability to translate ENDF-6 formatted files both to and from GND. This capability has already been used to create an unofficial release of the ENDF/B-VII.1 library in the GND/XML format. The resulting GND/XML files are available for testing and review at [http://www.nndc.bnl.gov/exfor/endfb7.1\\_gndfiles.jsp](http://www.nndc.bnl.gov/exfor/endfb7.1_gndfiles.jsp). This capability also opens up the possibility of modifying data in GND format, either manually or using a package like FUDGE, and then retranslating the modified data back to ENDF-6. More details on using FUDGE to translate ENDF-6 files can be found in Sec. VIII.

We recognize that the international community has made substantial investments in representing, processing and using nuclear data in the ENDF format. Also, many world wide application codes are tied to the usage of ENDF-6 formatted data (from ENDF/B-VII.1, JENDL 4.0, JEFF 3.1, CENDL, RUSFOND, etc.) via processed data using the NJOY code [5]. For this reason, we are working closely with key organizational groups such as CSEWG and WPEC as we prepare the transition to GND. To smooth this transition, we plan to make the next versions of ENDF/B, and other major libraries released in ENDF-6 format, also available in GND. Furthermore, we expect other processing codes including NJOY to begin supporting the processing of GND-formatted evaluated data.

This paper starts with a high-level overview of the new GND structure, which is proposed as a modern replacement for the ENDF and ENDL formats. We then explore the proposed new structure, starting with high-level elements such as the resonances and reaction elements, and working down to more specific quantities like cross sections, products, distributions and multiplicities, as well as the general-purpose data containers that are used within each of these data types. We discuss the implementation of GND in the FUDGE code, including tools for translating data back and forth between ENDF and GND. We will discuss the progress that has been made in developing several tools meant for use along-side GND, including a new database for storing nuclear structure information, and fast access routines for reading GND data. We also discuss the status of a new subgroup (number 38) of the Working Party for Evaluation Cooperation (WPEC), that will oversee the process of creating an international standard for storing nuclear data based on GND.

## II. OVERVIEW OF GND

GND (like its predecessors ENDF and ENDL) is designed to store evaluated nuclear reaction data produced by the Nuclear Data community. These evaluations are

TABLE II: High-level overview of the nested hierarchy making up the GND structure. Each item in the list corresponds to an element in GND. The top-level reactionSuite element contains other elements such as styles, documentations and reaction. Each of these in turn may contain other elements and/or datasets.

---



---

• reactionSuite (specifying the projectile and target)
– styles
– documentations
– particles
– resonances (optional)
* resolved
* unresolved
– reaction 1
* documentation (optional)
* crossSection
* outputChannel
· product 1
· product 2
· ... (each product contains distributions, multiplicity, etc.)
– reaction 2
– ...

---



---

produced by integrating experimental results and model calculations together into an evaluation containing a single set of recommended values for reaction cross sections and outgoing particle distributions. The evaluated data can then be processed into forms suitable for modeling the transport of nuclear particles through materials using Monte Carlo or deterministic transport codes.

The fundamental building block of an evaluation is the ‘reaction’ element, which consists of an input channel, output channel and a cross section. As with ENDF, GND only considers two-body input channels involving one projectile and target. In GND, all reactions with the same input channel are grouped together in an element called a ‘reactionSuite’ (for example, a reactionSuite might consist of all reactions incurred by a neutron incident on  $^{238}\text{U}$ ).

The output channel for each reaction consists of its set of outgoing particles, or ‘product’ elements. Each product is described by probability distributions for the outgoing angle and/or outgoing energy and by an associated multiplicity (the number of products corresponding to that distribution).

In addition to the list of reactions, the reactionSuite also contains other elements, such as documentation and a list of resonance parameters, that apply to more than one reaction.

The major elements of the reactionSuite hierarchy are shown in Table II. This hierarchy mirrors the underlying nuclear physics and can be implemented easily in

any meta-language capable of storing a nested hierarchy. Each element in this hierarchy will be described in more detail in Sec. V.

### III. NAMING CONVENTION

Unlike ENDF and ENDL that use integers to represent names for objects (e.g., particles via ZA and reactions via MT), GND uses strings. Furthermore, both ENDF and ENDL use different integers for the same isotope, depending on its use. As example, an isotope in ENDF-6 is identified both by an integer ‘material’ number and a second value for its ‘ZA’ (both related to it being developed when computer technology was in its infant stage). In ENDF-6 formatted files,  $^{16}\text{O}$  is represented as the integer 825 for its material name and 8016 ( $1000 \times Z + A$ ) for its ZA name (this MAT convention is described in Sec. 0.4.1 of [1]). GND has only one name for each object.

This section describes the GND naming convention for particles and reactions.

#### A. Particle and product naming

In GND, the name of an isotope is a combination of its atomic symbol (S) and nucleon number (#) as ‘S#’. For a nucleus in excited level with level index L, the qualifier ‘\_eL’ is added. For example,  $^{16}\text{O}$ ,  $^{239}\text{Pu}$  and  $^{239}\text{Pu}$  in its fifth excited state are labeled ‘O16’, ‘Pu239’ and ‘Pu239\_e5’ respectively. If a nucleus is in an undefined excitation level, typically considered to be the continuum, the excited state suffix is ‘\_c’ (e.g., ‘Pu239\_c’ for the continuum (n,n’) reaction, ENDF MT 91, for target  $^{239}\text{Pu}$ ).

Aside from isotopes and their excited states, only two other particles are currently required to translate the incident neutron, charged particle and gamma sub-libraries from ENDF and ENDL into GND. These particles are the neutron named ‘n’ and the gamma named ‘gamma’.

In GND an outgoing particle is called a product. A product is more than a particle in that a product also contains a multiplicity and an outgoing energy-angular distribution. The name for a product is taken from the name of its particle and qualified by its multiplicity if the multiplicity is other than one. For a product with an integer multiplicity N, other than one, the string “[multiplicity:N]” is appended to the name of its particle (e.g., “n[multiplicity:4]” for a neutron with multiplicity four). In GND there are three cases where the multiplicity of a product is not an integer: 1) neutrons from fission, 2) gammas in general and 3) products from the ‘sum of all remaining reactions’ (i.e., the GND equivalent of MT 5 in ENDF). In these cases, the string “[multiplicity:energyDependent]” is added to the particle name, indicating that the product contains an element named ‘multiplicity’ (see Section V F 2).

#### B. Reactions

Rather than the ENDF MT numbers (or the similar C and S numbers of ENDL format of LLNL), GND uses the list of particles in the input and output channels as they are stored in the GND structure to identify a reaction. This section describes how channels are represented in GND and how the representations are diagrammatically expressed.

In GND there are four objects that compose a reaction and make it distinct from other reactions: 1) the input channel, 2) the output channel, 3) the ‘process’ attribute for the reaction and 4) decay channels.

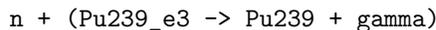
Diagrammatically, the input channel is the name of the input particles separated by the “+” character. As example, for a neutron projectile hitting the target  $^{239}\text{Pu}$ , the input channel is shown diagrammatically as “n + Pu239”.

For a simple output channel where none of the products decay, the output channel is diagrammatically given as the name of the products separated by the ‘+’ character. (Note, this is the list of product names and not the outgoing particle names, see Section III A.) As example, the output channel for the reaction  $^{239}\text{Pu}(n,3n)^{237}\text{Pu}$  can be shown as “n[multiplicity:3] + Pu237” indicating that the GND structure for this reaction contains two products, one representing the three outgoing neutrons with a single distribution and one for the outgoing  $^{237}\text{Pu}$ . If the evaluator wishes to represent each neutron with a different distribution, then the output channel will contain four products, one for each of the three neutrons and one for  $^{237}\text{Pu}$ . In this latter case, the output channel is diagrammatically shown as “n + n + n + Pu237”.

For some reactions the list of outgoing particles is insufficient to uniquely identify the reaction. For these reactions the process attribute must be used to break the degeneracy. Diagrammatically, the process attribute is given at the end of the output string as “[PROCESS]” where PROCESS is the name given to break the degeneracy. The output string and the process string are separated by the space character (i.e., “ ”). For example, nuclear reaction modeling codes (such as EMPIRE [6] and TALYS [7]) are able to differentiate between shape elastic and compound elastic reactions. ENDF-6 does not allow listing these reactions separately, but in GND they can both be identified using a reaction ‘process’ qualifier as “n + Pu239 [shape elastic]” and “n + Pu239 [compound elastic]” respectively. Separating these (and other) physical reaction processes will be advantageous to evaluators. For example, when performing actinide evaluations, one must often adjust the fission cross section obtained from the modeling code in order to fit the experimental data. To preserve the reaction cross section from the model, other reaction channels including compound elastic must be rescaled. Without separate reaction processes, the shape elastic (which does not contribute to the reaction cross section) cannot be separated from the compound elastic.

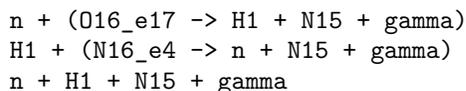
For some reactions one or more of the products de-

cays. The most common example of this is a two-body output channel where the heavy product is in an excited state which subsequently decays via gamma emission. Schematically, in GND a product that decays and its decay products are contained within the '(' and ')' characters with a '->' separating the product that decays from its decay products. As example, in the output channel "n + <sup>239e3</sup>Pu" the <sup>239e3</sup>Pu will decay to its ground state. This reaction is schematically given as



which indicates that the output channel is initially a two-body channel with a neutron and <sup>239e3</sup>Pu as products with the <sup>239e3</sup>Pu immediately decaying to <sup>239</sup>Pu and gammas. For this reaction, the <sup>239</sup>Pu and gammas are listed in the decay channel of <sup>239</sup>Pu.

In ENDF, the reaction (n,n p), MT 28, cannot distinguish whether the neutron or proton is emitted first, or both at the same time. In GND these reactions are distinct, and all can be present simultaneously. As example, for a neutron hitting <sup>16</sup>O, the following output channels would be summed into one MT 28 channel in ENDF



while they are all distinct in GND.

The Q-value for an output channel is calculated from the mass differences between the input channel particles and the output channel products. That is, the Q-value for the output channel does not use the masses of the products in any decay channel. Decay channels also have Q-values which are calculated from the mass differences between their parent product and the immediate decay products. For the prior (n,np) example, the reaction producing <sup>16e4</sup>N has a Q-value for the output channel of about -12.98 MeV (the sum of the masses of the neutron and <sup>16</sup>O minus the sum of the masses of the proton and <sup>16e4</sup>N) while the decay channel for <sup>16e4</sup>N has a Q-value of about 0.86 MeV.

Fission reactions also require extra information in the reaction qualifier. In GND, total fission producing one prompt neutron, six delayed neutrons, and gammas is labeled as "n[multiplicity:'energyDependent', emissionMode: 'prompt'] + n[emissionMode:'6 delayed'] + gamma [total fission]". The list of outgoing particles is complex due to the nature of fission, so the extra reaction qualifier "[total fission]" is added to help uniquely identify the reaction.

Table III shows examples of some common reaction identifiers in GND, along with their MT equivalents where possible.

#### IV. REPRESENTATIONS OF DATA IN GND

In order for GND to store evaluated data and derived data (suitable for plotting and transport codes) simul-

TABLE III: Example of reaction labels in GND. ENDF MT numbers are listed when possible, but some GND reactions have no MT equivalent.

GND reaction label		ENDF MT
n + Pu239	→ n + Pu239	2
n + Pu239	→ n + Pu239 [compound elastic]	
n + Pu239	→ n[multiplicity:'2'] + Pu238	16
n + Pu239	→ n[multiplicity:'3'] + Pu237_e1	
n + Pu239	→ n + Pu239_e1	51
n + Pu239_m1	→ n + Pu239_c	91
n + Pu239	→ Pu240 + gamma	102
n + Pu239	→ Pu240_e1 + gamma	
C12 + Pu239	→ C12_e2 + Pu239_e1	
n + Be7	→ (Be8 → He4[multiplicity:'2'])	

taneously, a quantity (e.g., cross section) must be able to hold various representations of its data concurrently. GND defines two types of representation containers. One is called 'form' and the other is called 'component'. In this section we discuss the usage of 'form' and 'component' in GND.

#### A. Forms

One of the goals for GND is to enable the storing of more than one representation of a quantity simultaneously. For example, an evaluator might represent a cross section as a combination of resonance parameters with a pointwise background (in ENDF this is represented with MF 2 and 3 data). The resonance parameters can be reconstructed to a pointwise representation and added to the background to achieve a fully pointwise representation for plotting, comparison with other cross sections, processing and use in continuous-energy Monte Carlo transport codes. The cross section might also be grouped into a standard set of energy bins for use with transport codes. GND allows all three of these representations of the cross section to be stored in a file simultaneously. This is different from ENDF, where cross section data can be stored as a combination of resonance parameters (MF 2) and background data (MF 3) or as resonance parameters (MF 2) and reconstructed pointwise data (MF 3), but not both simultaneously.

In GND, each representation of a quantity is called a 'form'. For the crossSection example above, the three forms are stored inside the crossSection element as schematically shown in Table IV. In addition to cross sections, examples of other quantities which have forms are multiplicity, and distribution components (see Sec. IV B).

While all the different forms are valid representations of a quantity, they are not equivalent. In the example of Table IV the 'resonancesWithBackground' form contains the original data, from which the pointwise and grouped forms are derived. GND introduces the concept of 'nativeData' to identify the original data form (that is, the

TABLE IV: Overview of a crossSection data element, highlighting different forms in which a cross section may be represented. The nativeData attribute indicates that the resonancesWithBackground form was originally entered by the evaluator, and that the pointwise and grouped forms are derived from resonancesWithBackground. Throughout this paper, GND attributes such as ‘nativeData’ will be italicized.

---



---

```

• reactionSuite
  - reaction 1
    * crossSection nativeData = "resonancesWithBackground"
      · form: resonancesWithBackground
      · form: pointwise
      · form: grouped
    * outputChannel ...
  - reaction 2 ...

```

---



---

form inputted by the evaluator). All quantities (i.e., elements) that contain forms have a nativeData attribute. The nativeData attribute in the crossSection element example of Table IV indicates that the ‘resonancesWithBackground’ is the form originally chosen by the evaluator, and that the ‘pointwise’ and ‘grouped’ forms are derived from it.

## B. Components

In order to store evaluated and processed distribution data simultaneously for each product, GND introduces the concept of a ‘component’ for distributions.

In general, probability distributions for outgoing products are described by the double-differential distribution

$$\frac{P(E', \mu|E)}{2\pi} = P(E', \Omega|E) = \frac{1}{\sigma(E)} \frac{\partial^2 \sigma(E, E', \Omega)}{\partial E' \partial \Omega} \quad (1)$$

where  $E$  is the projectile energy,  $E'$  is the product energy,  $\Omega$  is the directional solid angle,  $\mu$  is the cosine of the angle between the projectile and product velocities, and  $\sigma(E)$  and  $\partial^2 \sigma(E, E', \mu)/\partial E' \partial \Omega$  are the integrated and double-differential cross sections, respectively. (One exception to the general distribution described by Eq. 1 that appears in ENDF is Coulomb scattering, MF 6 with LAW 5. Coulomb scattering is handled by GND, but will not be described here as it is beyond the scope of this article). There are several ways to represent the data for  $P(E', \mu|E)$ . For example, it may be stored as pointwise double-differential data or as Legendre coefficients  $C_l(E, E')$  defined as

$$C_l(E, E') = \int_{-1}^1 d\mu P(E', \mu|E) P_l(\mu) \quad (2)$$

where  $P_l(\mu)$  is the Legendre polynomial of degree  $l$ .

As another example, for a two-body output channel only the center-of-mass angular data  $P(\mu|E)$  are needed since the outgoing energy can be derived from the angle (i.e.,  $P(E', \mu|E) = P(\mu|E)\delta(E' - E'(E, \mu))$ ). In GND,  $P(\mu|E)$  is called the ‘angular’ component. For processing or visualization, however, this angular distribution data may be expanded into full double-differential distribution data  $P(E', \mu|E)$  as a function of both energy and angle (‘energy-angular’ component). The two types of distribution (‘angular’ and ‘energy-angular’) are each a ‘component’ of the distribution.

As with forms, only one component for each distribution is the original component specified by the evaluator; the other components are derived from it. Hence, a distribution also requires a ‘nativeData’ attribute to indicate which component was originally chosen by the evaluator.

Each component of a distribution may contain its data in various forms. For example, for a two-body output channel, the angular data can be stored in pointwise form ( $P(\mu|E)$ ) and/or Legendre form ( $C_l(E) = \int_{-1}^1 d\mu P(\mu|E) P_l(\mu)$ ). An example of a possible distribution for a two-body output channel with various representation of its data is shown in Table V. In this example, the evaluator inputted the distribution as an angular component in Legendre form (i.e.,  $C_l(E)$ ). Later, an angular pointwise representation of the angular/Legendre data was generated from the Legendre coefficients, using enough points to preserve the original behavior to a user-supplied tolerance. Finally, a Legendre/grouped component/form, suitable for use in deterministic transport codes, was generated. All three of these representations of the data, angular/Legendre, angular/pointwise and Legendre/grouped, can be stored simultaneously in GND.

TABLE V: An example of a distribution for a product of a two-body output channel. The distribution has two components (in bold), ‘angular’ and ‘Legendre’. Each component contains one or more forms. The ‘nativeData’ attributes indicate that the Legendre form of the angular component contains the original data inputted by the evaluator.

---



---

```

• distributions nativeData="angular"
  - angular nativeData="Legendre"
    * Legendre ...
    * pointwise ...
  - Legendre
    * grouped ...

```

---



---

Table VI shows the ENDF MF, LAW and LANG equivalent for the components/forms listed in Table V.

Using the concepts of form, component and nativeData, GND can contain one or more derived representations of a quantity, without overwriting the original data entered by the evaluator.

TABLE VI: ENDF equivalents for the components in Table V.

---



---

• <b>angular</b> $P(\mu E)$
– Legendre ENDF MF 6, LAW 2, LANG 0
– pointwise ENDF MF 6, LAW 2, LANG 12
• <b>Legendre</b> $C_i(E, E')$
– grouped ENDF MF 6, LAW 1, LANG 1

---



---

### C. Links

Links are an important part of GND, allowing the evaluator to refer to other elements within the file or even to elements in external files or databases. Examples of data requiring links include:

- Distributions for one reaction product may be treated as the recoil from another product, requiring a link to the other product.
- Production cross sections may be listed as an energy-dependent multiple of another cross section, requiring a link to the other cross section.
- Covariances are stored in a separate file from the quantities they correlate (as described in Sec. V H). Links are necessary to associate the covariance with the correct data.

In GND, links are listed using the ‘xPath’ syntax described in [8]. This syntax is similar to paths used in the Unix filesystem, supporting both absolute and relative links. For example, an absolute xPath link to the resonances element at the top level of a reactionSuite is listed as ‘xlink:href=“/reactionSuite/resonances”’.

## V. DETAILS OF THE GND STRUCTURE

The reader has now encountered a high-level overview of the GND reactionSuite, as well as some details of naming conventions and the use of multiple forms and components to describe the data. More details on the elements appearing inside the reactionSuite are described in this section.

The GND structure begins with a reactionSuite element, and all other data in the structure are nested within this element (the label ‘reactionSuite’ is used rather than ‘reaction suite’ due to the constraints of XML and other meta-languages, which do not allow white spaces inside the label of an element or attribute. For the same reason, labels like ‘cross section’ and ‘Legendre Pointwise’ become ‘crossSection’ and ‘LegendrePointwise’). The reactionSuite is defined by a combination of a projectile

(e.g., neutron, proton or gamma), with a target (e.g., nucleus in ground or excited state). It also contains a format version number, in order to support future additions or changes to the GND structure. The major elements of the reactionSuite are described next.

### A. The ‘style’ element

The first required element encountered inside the reactionSuite is named ‘styles’. This element contains one or more ‘style’ elements, each of which corresponds to the evaluated or/and processed data contained in the GND structure. For example, a GND file may contain both evaluated and grouped deterministic data. Each of these representations of the data must have a corresponding style element that contains information specific to that representation. For evaluated data, information such as library version number and names of evaluators are stored in its style element. For grouped deterministic data, information about the group energy boundaries is also stored inside the style element.

TABLE VII: Styles and some of their attributes currently allowed in GND.

	Name	Description
Styles	Evaluated	Original data from evaluator
	Processed	Deterministic Monte Carlo
Attributes	Library	ENDF/B, for example
	Version	VII.1, for example

### B. The ‘documentations’ element

GND also requires a ‘documentations’ element, that in turn contains one or more sections of documentation. These should contain a high-level overview of the file, including author and contact information, procedures and codes used in creating the file, and other references. Multiple sections can be used to store different forms of documentation for different media (e.g., one form for writing back to ENDF and another for display on the web).

Reaction-specific information can be stored in the documentations section, but GND also makes room for storing reaction-specific documentation inside the corresponding reaction element, which should be the best practice for future evaluations (see Sec. V E).

### C. The ‘particles’ element

Particle information like mass, proton number and nuclear levels are needed for evaluations and transport codes. Unlike ENDF and ENDL, which repeat these data many times in different sections (e.g., the mass for the

target is replicated many times in an ENDF file, and often these values differ), GND stores each datum only once. Particle information are stored inside the ‘particles’ element. The particles element contains a list of ‘particle’ elements where data for each particle are stored. A schematic of a particle element is shown in Table VIII.

TABLE VIII: Example of the structure for storing particle information such as masses, excited levels and decays. In the future, the particles element may contain a link to an external particle database.

---



---

- particles
  - particle *name*="F19" *mass*="18.998204 amu"
    - \* level *name*="F19\_e1", *index*=1, *energy*="109.9 keV"
      - gamma *finalLevel*="F19\_e0" *probability*="1.0"
    - \* level *name*="F19\_e2" *energy*="1.97e5 eV"
    - \* ...
  - particle *name*="F20" *mass*="19.9999 amu"
  - ...

---



---

When a suitable structure for an external particle database is developed, the GND design will allow a reactionSuite to point to an external particle database so that a collection of reactionSuites (different targets and potentially different projectiles) can all be built using consistent particle information. This option will likely not be used in the near term since many older evaluations in nuclear data libraries have been built using different values for masses. In the future, however, this will provide a way to keep nuclear data consistent and to avoid problems with inconsistent Q-values and thresholds. A discussion of progress towards defining a structure for a particle database is given in Sec. IX F.

#### D. ‘resonances’ element

Resonance parameters are a compact way to store the low-energy cross sections for multiple reactions (and can in principle also be used to calculate distributions). Since they apply to more than one reaction, resonances are stored outside of any reaction elements. In the rapidly fluctuating resolved resonance region, each resonance can be described by its energy, spin, parity and partial widths (one width for each open channel). In the intermediate or ‘unresolved’ resonance region where resonances increasing overlap, average resonance parameters may be used to represent the data. In both regions, the resonance parameters can be used to reconstruct pointwise, linearly-interpolable data.

GND follows the example of the ENDF format by permitting the use of several different resonance parameter formalisms in both the resolved and unresolved resonance

regions. In the resolved region, resonances may be represented using various approximations to the general R-Matrix formalism [9, 10], including Single- and Multi-Level Breit Wigner, Reich-Moore and a limited R-Matrix approach. In the unresolved region, only the Single-Level Breit-Wigner approximation is allowed (similar to ENDF). In each case, the resonance parameters are stored using the ‘table’ data container described in Sec. VI F.

The package FUDGE includes a function for reconstructing the cross section from resonance parameters stored in GND. This code will be described further in Sec. VII A.

#### E. ‘reaction’ element

Most of the data in a reactionSuite will usually reside within the ‘reaction’ elements. In GND, there is one reaction element for each distinct output channel. A distinct output channel is any channel for which the physical process or order of products emitted can be distinguished, and which the evaluator has decided to demarcate. Here are several examples.

1. For elastic scattering, the evaluator may provide a single elastic scattering reaction, or she may decide to separate shape and compound elastic scattering into two reaction elements (one with process attribute "shape elastic" and the other with "compound elastic").
2. In ENDF the reaction ‘n + N14 → n + H1 + C13’ does not distinguish whether the ‘n’ or the ‘H1’ is emitted first (or whether both are emitted at the same time). If this reaction is actually the two body reaction ‘n + N14 → H1 + C14’ followed by the two body decay of the ‘C14’ to ‘n + C13’ then only the center-of-mass angular distributions are needed, which can save data storage space.
3. The reaction<sup>1</sup> ‘H3 + H3 → 2n + H4’ could be represented with three distinct output channels which we label as ‘2n + H4’, ‘n + (H5 → n + H4)’ and ‘n + (H5\_e1 → n + H4)’. In GND each of these can be listed as separate reaction elements.

Each reaction element contains the cross section, products, distributions, etc. for that reaction. This is opposite to the convention in the ENDF format, where all cross section data for all reactions are grouped together in the MF 3 (and possibly MF 2) file, all angular distribution information are grouped in MF 4 or MF 6, and so on. The ENDF approach scatters data for a single reaction across the entire file, and may lead nuclear data evaluators to address reaction data as separate components

---

[1] For brevity ‘n[multiplicity="2”]’ is written as ‘2n’.

rather than considering them all as part of a whole. GND departs from the ENDF convention, storing all reaction data together in a more unified and physical layout.

GND divides reactions into two types, complete and incomplete. A complete reaction is one for which all products (with the possible exception of gammas) are explicitly listed in the output channel. For these reactions, the checking codes (see Section VII C 2) can run additional tests to ensure that quantities like neutron and proton number are conserved. (If a reaction contains a weak-decay, then proton number need not be conserved, but as long as the change can be determined and the correction balances, the reaction is still considered complete.) Reactions that do not explicitly list all products are called ‘incomplete’. Current evaluated libraries contain two types of incomplete reactions: fission, which is missing protons and neutrons from the outputChannel since fission products are not explicitly listed, and the ‘sum of all remaining reactions’ (corresponding to MT 5 in the ENDF format), which generally has too many protons and neutrons in the outputChannel since products from multiple different reactions have been lumped together.

### 1. The ‘crossSection’ element

Every reaction element in a GND structure must contain a ‘crossSection’ element. The cross section ( $\sigma(E)$  where  $E$  is the projectile incident energy) may be represented in several different ways as shown in Table IX.

Each of the representations in Table IX is a ‘form’. As discussed in Sec. IV, multiple forms of the same data can be stored simultaneously. In that case, the ‘nativeData’ attribute clarifies which data form was chosen by the original evaluator, and which forms are derived. An example of (XML-formatted) GND cross section data is shown in Fig. 1.

TABLE IX: Allowed forms for storing a cross section in GND.

Form	
pointwise	single interpolation region, using the ‘XYs’ data container discussed in Sec. VI
piecewise	multiple interpolation regions, using the ‘regionsXYs’ container
resonances- WithBackground	has a link to a resonance element, and a pointwise or piecewise form that should be added to reconstructed resonances (equivalent to ENDF MF 2 and 3)
weightedPointwise	cross section is an energy-dependent multiple of the cross section from another reaction (this occurs when data is translated from ENDF MF 9)
grouped	average cross section over energy intervals

Some cross section forms require the use of links. For example, the resonancesWithBackground form seen in Fig. 1 uses a link to point to the resonance parameters that must be reconstructed and added to the background. In that example, the full path to the link is given (starting with the top-level reactionSuite element). This link points to another resource within the same file, but links could also be used to point to external files.

### 2. The ‘outputChannel’

In addition to the cross section, a complete description of a nuclear reaction requires a list of outgoing products. In GND, these products are collected together inside the ‘outputChannel’ element. The outputChannel contains a Q-value as well as a list of products.

Outgoing reaction channels fall into two categories: two-body and uncorrelated bodies. If only two particles are initially produced, and the excitation energy of each product is known, the reaction is called ‘two-body’. In this case, only the angular distribution in the center-of-mass frame for one product must be given; the angular distribution for the other product can be obtained by symmetry, and the energy of both products can be calculated from their angle in the center-of-mass frame. Reactions are called ‘uncorrelated’ if they involve more than two primary products or if the excitation energies are not known. For ‘uncorrelated’ reactions, the probability distributions for each product must be specified as a function of incident energy, outgoing energy and outgoing angle. That is,  $P(E', \mu|E)$  must be given for each product, and the distribution for each product is uncorrelated with the distributions given for the other products.

TABLE X: Genres for outputChannel and decayChannel elements (see Sec. V F 3).

Genre	Description
two-body	only two products are emitted per channel, the products are correlated, and only the center-of-mass angular distribution is needed in order to calculate the double-differential distribution
uncorrelated	the products are uncorrelated from each other, and a complete double-differential distribution is required in order to describe each product

GND introduces the concept of ‘genre’ in order to specify whether the output channel can be treated as ‘two-body’ or ‘uncorrelated’. The allowed genres for an outputChannel in GND are summarized in Table X.

In addition to the genres listed in Table X, a special channel genre called the ‘sumOfRemainingOutputChannels’ is used to store the equivalent of ENDF MT 5. This genre is used when many outgoing reactions are possible and are difficult to separate with experiment (often

```

<crossSection nativeData="resonancesWithBackground">

  <resonancesWithBackground>
    <resonanceRegion xlink:type="simple" xlink:href="/reactionSuite/resonances"/>
    <piecewise xData="regionsXYs">
      <axes>
        <axis index="0" label="energy_in" unit="eV" interpolation="byRegion,byRegion" frame="lab"/>
        <axis index="1" label="crossSection" unit="b" frame="lab"/></axes>
      <region index="0" length="2" accuracy="0.001">
        <interpolationAxes index="0" interpolation="linear,linear"/>
        <data> 1e-5 0 3e4 0</data></region>
      <region index="1" length="283" accuracy="0.001">
        <interpolationAxes index="0" interpolation="linear,linear"/>
        <data> 3e4 1.63726 ... 2e7 2.360617</data></region>
    </piecewise>
  </resonancesWithBackground>

  <pointwise xData="XYs" length="268399" accuracy="0.01">
    <axes>
      <axis index="0" label="energy_in" unit="eV" interpolation="linear,linear" frame="lab"/>
      <axis index="1" label="crossSection" unit="b" frame="lab"/></axes>
    <data> 1e-5 36525.3468075 1.2626001e-5 32505.91011 ... 2e7 2.360617</data>
  </pointwise>

</crossSection>

```

FIG. 1: Sample cross section data in GND/XML, containing two different forms of the cross section data. The resonancesWithBackground form contains the original evaluated data, and the pointwise form was derived from it (by reconstructing resonances and then adding the piecewise background).

at higher incident energies). The sumOfRemainingOutputChannels applies only to reactions, not to decays.

## F. ‘product’ elements

Product distributions and multiplicities are stored inside the ‘product’ element in GND. The outline of a product element is given in Table XI:

TABLE XI: Organization of the ‘product’ element in GND.

- 
- product (including particle identifier)
    - distributions
      - \* components
        - forms
    - multiplicity
    - decayChannel (optional)
    - average energy and/or momentum (optional, derived data)
- 

The product element first identifies the outgoing particle (‘n’, ‘gamma’, ‘Pu239’ etc.), then lists distributions and multiplicity for that particle.

### 1. ‘distributions’

Each product must contain a ‘distributions’ element. If no distribution data are available, the element must still be present, with the nativeData attribute set to ‘none’.

As described in Sec. IV, a distribution element contains one or more component elements, each of which contains one or more form elements. Table XII lists some of the available components and their forms. The uncorrelated distribution component (not to be confused with the uncorrelated output channel genre) is a special case that must contain both an angular and an energy component in order to fully describe the distribution (similar to the combination of MF 4 and 5 in ENDF-6). Currently, instances of the energy component appear only within an uncorrelated component.

### 2. ‘multiplicity’

In addition to distributions, each product in GND requires a multiplicity. For many simple reactions such as (n,2n), the multiplicity is constant and may be given as an attribute of the product element. In other cases such as fission and capture, the product multiplicity varies with incident energy. In this latter case, the average multiplicity (generally non-integer) is listed as a function of incident energy and the data are stored using the XYs

TABLE XII: Partial list of distribution components and forms currently allowed in GND.

Component	Form	ENDF equivalent
angular	pointwise	MF4 LTT2
	Legendre	MF4 LTT1
	isotropic	MF4 LTT0
	recoil	MF6 LAW4
energy	pointwise	MF5 LF1
	Evaporation Spectrum	MF5 LF9
	Maxwellian	MF5 LF7
	Watt Spectrum	MF5 LF11
	Madland-Nix	MF5 LF12
	N-Body Phase Space	MF6 LAW6
energy-angular	Kalbach-Mann	MF6 LAW1 LANG2
angular-energy	pointwise	MF6 LAW7
uncorrelated	combination of 'angular' and 'energy'	MF4 and MF5

data container (described in Sec. VI).

### 3. Other product data

Some optional data types may appear inside a product element. For example, if the product is subject to radioactive decay, that information can be stored in a 'decayChannel' element. The decayChannel is very similar to the outputChannel (discussed in Sec. V E 2). It contains a Q-value, a genre (indicating whether the decay is two-body or uncorrelated bodies), and a list of decay products including distributions and multiplicities.

A product element may also contain average energy and momentum data. These data can be calculated for all products that contain distribution data. The average energy information provides a simple way to test that the evaluation conserves energy (discussed further in Sec. VII C). Like cross section and multiplicity, average energy and momentum are one-dimensional functions and are stored using the XYs data container.

After processing, a GND file may also contain transfer matrices for each product. These transport matrices are used in deterministic transport codes.

## G. Other top level elements

In GND, a reaction element can only store a distinct reaction. That is, each reaction element stores a fraction of the total cross section, and summing up all the reactions shall reproduce the total cross section. However, evaluators often need to store cross sections that represent sums over more than one distinct reaction channels, or that represent a portion of one distinct channel. The GND structure supports various sum and partial quantities via the 'summedReaction', 'fissionComponent', 'partialGammaProduction' and 'production' elements, which are stored under the 'reactionSuite' element.

The summedReaction element allows nuclear data evaluators to supply a cross section for a sum of various distinct reactions. For example, ENDF MT 1 contains the total cross section, summed over all distinct reactions, and MT 4 contains the sum over all inelastic reactions. The summedReaction element contains a list of the reactions that are being summed over, a cross section, and a Q-value. Unlike the reaction element, the summedReaction contains no output channel or description of products.

The summedReaction element is in principle redundant, since the same cross section could be computed by summing up the cross section of all the constituent reactions. However, these data are still important since summed cross sections such as total may have been measured experimentally even though constituent reactions have not.

The 'fissionComponent' element allows for subdividing fission into its component reactions. Physically, fission consists of many different reactions depending on the combination of fission products. For simplicity, these are grouped together into a few categories: 'first-chance', where the compound nucleus undergoes fission, 'second-chance' where the compound nucleus first emits a neutron and then fissions, and so on. Evaluations often contain a description of these separate fission components along with a description of their total sum. However, outgoing product distributions are usually only available for the total fission, not for its components. Rather than listing first and second chance as reaction elements and total fission as a summedReaction element, GND stores the total fission as a reaction element and uses a fissionComponent to store first-chance, second-chance and so on. The fissionComponent element is nearly identical to a reaction element, containing a cross section and an output channel (in some cases the output channel will contain distributions, although the relationship between this data and the distributions listed for the total fission reaction is unclear). In addition, it contains a 'fissionGenre' attribute, which may be 'firstChance', 'secondChance' and so on. Summing up the fissionComponent cross sections should reproduce the total fission cross section.

Nuclear data evaluations sometimes contain gammas whose distinct reaction is unknown. Since these gammas cannot be associated with a particular reaction, they are instead listed together in the 'partialGammaProduction' reaction in GND. This element (corresponding to ENDF MT 3) lists the cross section, multiplicity and outgoing distributions for all gammas not associated with any distinct reactions.

Finally, the 'production' element is used to store the cross section for producing radioactive products (corresponding to ENDF MFs 8, 9 and 10). The production element identifies the radioactive product, and contains a cross section and a Q-value, but no output channel.

This section has introduced several new elements that may appear in a reactionSuite. For clarity, table XIII summarizes all the elements that may appear at the top

TABLE XIII: List of all elements that are allowed at the top level within a reactionSuite. Some elements are optional; others may appear more than once.

element	number allowed
styles	1
documentations	1
particles	1
resonances	0 or 1
reaction	1 or more
partialGammaProduction	0 or 1
summedReaction	0 or more
fissionComponent	0 or more
production	0 or more

level inside a reactionSuite. In the current definition of GND, the order of these elements is important. For example, all reaction elements must be listed before any partialGammaProduction or summedReaction elements.

## H. Covariances

In addition to the reactionSuite, GND defines a structure called the ‘covarianceSuite’, used for storing nuclear data covariances. The covarianceSuite is stored in a separate file from the reactionSuite and uses links to associate covariance data with the correct reaction data. This is different from the design of most of GND, which usually attempts to gather all data for a reaction in a single place. To understand the reason behind this departure, we explore the types of covariance data that are now becoming available in reaction libraries.

Recent releases of ENDF-formatted libraries have begun to include more covariance matrices. These include simple covariances between the evaluated cross section for a single reaction at two different incident energies, and also more complex covariances between different reactions or even between reactions involving different targets or projectiles. Together, all these individual covariance sections can be considered as rectangular blocks that make up a much larger covariance matrix. Covariances for one reaction at different energies are clustered around the diagonal of this larger matrix, while cross-terms between different reactions or between different materials lie away from the diagonal. The matrix may become even more complex in the future if correlations start to be given for different types of data (between angular correlations and reaction cross sections, for example). The ENDF-6 format allows for all these types of covariances, although some types are not yet used in any ENDF-VII.1 evaluation.

The complexity of covariance data, the fact that a covariance may couple more than one projectile/target, and the size required to store covariances are the main reasons for choosing to store covariances separately in GND. Also, since many applications of nuclear data do not use the covariance data, storing covariances separately means

that programs can save time and memory by not reading them in.

TABLE XIV: Organization of the covarianceSuite in GND. The covarianceSuite usually corresponds to the combination of one target and one projectile (like the reactionSuite). However, covariances may also be given between reactions for different targets and/or projectiles, which a covarianceSuite must also be able to handle. According to the current design of GND, all of the ‘cross-material’ covariances for a library should be stored together in a single multi-material covarianceSuite.

- covarianceSuite
  - styles
  - reactionSums (optional)
  - resonanceParameterCovariance (optional)
  - section 1
    - \* rowData
    - \* columnData
    - \* covarianceMatrix (or ‘mixed’, containing the sum of two or more covarianceMatrix elements)
  - section 2
  - ...

The structure of the covarianceSuite is seen in Table XIV. As in the reactionSuite, the ‘styles’ element is used to give library version information along with any supplemental information about processing. The ‘reactionSums’ section is necessary since a single covariance matrix may be given for the sum of several reactions, rather than for each reaction individually. This sum is defined in the reactionSums section (including a list of all the component reactions).

Fitting data with a model often results in a covariance matrix between the model parameters. GND supports storing these model parameter covariances. The resonanceParameterCovariance section is a special case.

The bulk of the covarianceSuite is made up of ‘section’ elements. Each of these sections represents one block of the full covariance matrix for this combination of target and projectile. Sections are identified by the data they correlate, that is, by their ‘rowData’ and ‘columnData’.

A sample GND covariance matrix is shown in Fig. 2. This example shows a typical symmetric matrix that stores the covariances between different incident energies for a single reaction. GND offers several choices for saving space when storing symmetric or sparse matrices. For more details, see Sec. VI F.

```

<covarianceMatrix type="relative">
  <axes>
    <axis index="0" label="row_energy_bounds" unit="eV"
      interpolation="linear,flat" length="7"> 1e-5 1.4e6 2e6 4e6 8e6 1.6e7 6e7</axis>
    <axis index="1" label="column_energy_bounds" unit="eV"
      interpolation="linear,flat" mirror_row_energy_bounds="true"/>
    <axis index="2" label="matrix_elements" unit=""/></axes>
  <matrix dimensions="6,6" form="symmetric" precision="6">
    3.671000e-05
    2.846000e-05 2.407000e-05
    1.920000e-05 1.914000e-05 1.907000e-05
    2.493000e-06 1.024000e-05 1.894000e-05 3.463000e-05
    -1.924000e-05 -1.325000e-06 1.877000e-05 5.504000e-05 1.022000e-04
    -3.276000e-05 -8.521000e-06 1.866000e-05 6.774000e-05 1.316000e-04 1.713000e-04</matrix>
</covarianceMatrix>

```

FIG. 2: Sample GND/XML covariance data. Energy group boundaries are stored inside the axes element, followed by the matrix elements. Since this matrix is symmetric, only the lower-diagonal portion of the matrix is stored (and energy boundaries are only required on one axis).

## VI. GENERAL-PURPOSE DATA CONTAINERS FOR GND

One of the main goals for GND (defined in Sec. I) is to define general-purpose data containers for storing various kinds of nuclear data. Fortunately, only a few fundamental data containers are necessary in order to store most nuclear reaction data. These containers are used in many places throughout GND. They include containers for one-, two- and three-dimensional functions, as well as tables and matrices.

### A. Axes element

In GND, containers for one-, two- or three-dimensional functions contain not only numerical data, but also information about each axis. Each axis contains ‘label’, ‘index’, ‘unit’ and ‘frame’ attributes. The label is a string for the name of that axis (e.g., “mu” for the  $\mu$  axis) and may be used for labeling the axis when the data are plotted. The index associates the axis with a column of data (that is, the first independent axis has `index=0` and so on). The unit describes the units for that axis (e.g., “eV” or “kg”), and the frame describes which frame, lab or center-of-mass, the data are given in.

Each independent axis also contains an interpolation attribute that is a string of the form “qualifier:independent,dependent”. The allowed values for each part of this string are listed in Table XV. A full description of each interpolation type and qualifier is available in Sec. 0.5.2 of the ENDF format manual [1].

In Table XV, the ‘flat’ and ‘chargedParticle’ interpolations are valid only for the dependent axis. The ‘byRegion’ interpolation indicates that a separate interpolation will be associated with each region of a piecewise function.

Sample axes information for angular distribution data

TABLE XV: Interpolations in GND are listed in a string of the form “qualifier:independent,dependent”. The values allowed for “qualifier”, “independent” and “dependent” are listed here.

	Options
qualifier	unitBase correspondingPoints
independent/dependent	linear log byRegion flat chargedParticle

might look like:

- axes
  - axis `index=0`, `label="energy_in"`, `unit="eV"`, `interpolation="linear,linear"`, `frame="lab"`
  - axis `index=1`, `label="mu"`, `unit=""`, `interpolation="linear,log"`, `frame="centerOfMass"`
  - axis `index=2`, `label="P(mu|energy_in)"`, `unit=""`, `frame="centerOfMass"`

For the “energy\_in” axis, the interpolation “linear,linear” is equivalent to ENDF INT 2, and the interpolation “linear,log” on the “mu” axis is equivalent to ENDF INT 4. As with ENDF, the interpolation for each independent axis defines how to interpolate while moving along that axis.

### B. One-dimensional function containers

In a nuclear data evaluation, it is common to store a one-dimensional function  $y(x)$  (e.g.,  $\sigma(E)$  or  $\bar{\nu}(E)$ ) as numerical data; that is, as a list of  $(x_i, y_i)$  pairs where

$x_i < x_{i+1}$ . Herein, this type of data will be called one-dimensional pointwise data. In GND, an element container for one-dimensional pointwise data has attributes named 'xData', 'length' and 'accuracy', and elements named 'axes' and 'data'. The value for the 'xData' attribute must be "XYs". The 'data' element contains the pointwise data. As example, cross section data evaluated at the two energy points (1e-5 eV, 1.21e+4 b) and (0.01 eV, 12.1 b) would be represented as

- pointwise `xData="XYs", length=2, accuracy=0.001`
  - axes
    - \* axis `index=0, label="energy_in", unit="eV", interpolation="log,log", frame="lab"`
    - \* axis `index=1, label="crossSection", unit="b", frame="lab"`
  - data: 1e-5 1.21e+4 0.01 12.1

The attribute 'accuracy' has no equivalent in ENDF or ENDL. This attribute is required whenever changing an interpolation or doing computations on the data, in order to preserve the original accuracy. For example, in the sample above the cross section uses log-log interpolation to represent the 1/E behavior of a cross section at low energy. If this cross section is converted to linear-linear interpolation by a processing code or plotting tool, extra points must be inserted between 1e-5 eV and 1e-2 eV in order to preserve the original 1/E behavior. The attribute 'accuracy=0.001' indicates that enough points must be added to ensure that a linear-linear representation of the data can be interpolated anywhere in this region, and still agree to 0.1% or better with the original log-log accuracy as specified by the evaluator.

The pointwise XYs dataset uses the same interpolation throughout its domain. Some data, however, are better described using different interpolations in different regions. GND defines a 'piecewise' XYs data container, called 'regionsXYs', which contains one XYs container for each interpolation region. All the regions share the same axes, but each region specifies its own interpolation. The data for all adjacent regions must overlap at exactly one point, that is, the last x-value of the lower region must equal the first x-value of the higher region. The y-values of these two points need not agree, which allows for a discontinuous step function.

### C. Two-dimensional function containers

Two-dimensional functions  $y(w, x)$  (with two independent axes ( $w$  and  $x$ ) and one dependent axis  $y(w, x)$ ) are used in GND for storing probability distributions such as the angular distribution,  $P(\mu|E)$ . Here, the probability may be isotropic at low incident energies but strongly angle-dependent at high incident energies. The most efficient way of storing these data is therefore to allow a

different list of  $\mu$ -values for each incident energy. Thus, in both ENDF and GND two-dimensional data are stored as a list of  $(w_i, y_i(x))$  pairs; that is, for a given  $w$  value (corresponding to incident energy in the example above), the data are given as a one-dimensional function with only as many points are necessary for that  $w$  value.

For this type of data, GND defines the W\_XYs element container which consists of a list of one-dimensional pointwise data (i.e., an XYs type element) and their associated  $w$  values by ascending  $w$ . An W\_XYs element contains the attribute 'xData' with the value "W\_XYs" as well as an axes element and a list of XYs elements each with a 'value' attribute. The 'value' attribute contains the numerical  $w$  value for the XYs element. A simple example of an angular distribution with data at incident energies 1e-05 eV and 1000.0 eV may look like:

- pointwise `xData="W_XYs"`
  - axes
    - \* axis `label="energy_in", unit="eV"`
    - \* axis `label="mu", unit=""`
    - \* axis `label="P(mu|energy_in) unit=""`
  - energy\_in `value=1e-05 index=0`
    - \* data -1 0.5 1 0.5
  - energy\_in `value=1000.0 index=1`
    - \* data -1 0.6 0 0.4 1 0.6

For brevity, attributes describing the frame, interpolation and accuracy have been omitted. Note, the XYs elements are named per the label of the  $w$  axis ("energy\_in" for this example). The 'W\_XYs' data container does not require that each XYs container comprise the same set of  $x$  values.

### D. Three-dimensional function containers

A three-dimensional function  $y(v, w, x)$  has three independent axes ( $v, w$  and  $x$ ) and a dependent axis  $y(v, w, x)$  (a nuclear data example is the double differential differential  $P(E', \mu|E)$ , see Eq. 1). For the same reasons described in Sec. VI C, both ENDF and GND store this three-dimensional data as a list of  $(v_i, y_i(w, x))$  where the two-dimensional data  $y_i(w, x)$  are stored as a list of  $(w_i, y_i(x))$  and contain only enough  $w$  values to adequately describe the distribution for  $v$ . For this type of data, GND defines the V\_W\_XYs container which consists of a list of W\_XYs containers and their associated  $v$  value in ascending  $v$ . A V\_W\_XYs element contains the attribute 'xData' with the value "V\_W\_XYs" as well as an axes element and a list of W\_XYs elements each with a 'value' attribute. The 'value' attribute contains the numerical  $v$  value for the W\_XYs element.

### E. Legendre containers

Some distributions representable with a `W_XYs` container can also be represented using a variant of a Legendre series. As example, the angular distribution  $P(\mu|E)$  can be expressed as  $P(\mu|E) = \sum_{l=0}^{\infty} C_l(E)P_l(\mu)$  where the  $C_l(E)$ 's are energy dependent Legendre coefficients. The GND container `W_XYs_LegendreSeries` is designed to hold `W_XYs` data where the  $x$  data represents an angle from 0 to  $\pi$  (e.g.,  $\mu = \cos(\theta)$  for  $-1 \leq \mu \leq 1$ ). This is very similar to `W_XYs` data except that the `XYs` elements can be represented as a Legendre series. Similarly, the `V_W_XYs` container has the associated Legendre container `V_W_XYs_LegendreSeries` for data where the inner most containers (the `XYs` containers) can be represented as a Legendre series. A nuclear data example of this would be a double differential of the form  $P(E', \mu|E) = \sum_{l=0}^{\infty} C_l(E, E')P_l(\mu)$ .

### F. Table and matrix containers

'`XYs`', '`W_XYs`' and '`V_W_XYs`' are general data containers for numerically representing functions of one, two and three dimensions, respectively. They provide an efficient way of storing much of the data that appears in a nuclear reaction evaluation. A few other data containers are still required, however, including the 'table' and the 'matrix'.

A GND table element contains attributes listing the number of rows and columns, followed by a list of column headers with a label and unit for each column, and finally a list of data points with length  $rows \times columns$ . Currently, every cell in the table must contain a numeric value, although "unknown" or "NAN" table values may also be supported in the future. Tables are used in GND for storing resonance data. A table is convenient for this purpose since the same model parameters must be given for each resonance.

Matrices are used in GND to store covariances and grouped deterministic transport data. A matrix requires the following attributes: number of 'rows' and 'columns', and 'form' which determines how the matrix is represented. The matrix forms that are currently allowed in GND are:

- 'diagonal', in which only diagonal elements must be stored
- 'symmetric', in which the lower-diagonal portion of the matrix is stored
- 'asymmetric', requiring the full matrix
- 'sparse\_symmetric', and
- 'sparse\_asymmetric', in which non-zero matrix elements are stored along with row and column indices.

These matrix forms all correspond to existing choices within ENDF (although we note that sparse matrices are currently only allowed in ENDF for storing resonance parameter covariances). An example of a symmetric matrix is shown below:

- matrix *rows=3 columns=3 form="symmetric"*
  - data
 

15.7
-0.82 3.21
1.42 -0.24 1.67

In the future, other matrix forms may be added in order to save space in GND. For example, if most of the eigenvalues of a matrix are zero or very small, GND could save space by only storing the large eigenvalues and corresponding eigenvectors. Space can also be saved by using a binary format to store compressed matrices.

The most common application of the 'matrix' data type is for representing covariances between reaction cross sections or distributions at different incident energies. These covariances must contain not only a matrix, but also a description of each axis. The matrix axes are similar to the axes for `XYs` datasets, except that in addition to the label, unit and frame attributes the covariance matrix axes also contain a list of energy boundaries.

In this section the most common data containers that appear in GND files were presented. These data containers are used repeatedly throughout GND files for storing all types of data that appear in nuclear reaction evaluations. In addition, these data containers have been designed to be general-purpose, so that in the future they could be used to store other types of nuclear data including experimental data and nuclear structure data. In that case, re-using general purpose data containers is expected to simplify the maintenance of all these nuclear data libraries.

## VII. ASSOCIATED TOOLS FOR GND

The GND format alone is not particularly useful without a suite of tools that can be used to interact with and manipulate the data. We have developed several code tools to enable users to create, test and display nuclear data. Additionally, we are developing access routines to enable the use of GND by transport codes and other applications. In this section, we will discuss progress towards building a set of associated tools for GND.

Many of the associated tools described here require the use of the code framework 'FUDGE' which is maintained by the Computational Nuclear Physics group at LLNL and is available for free download from the National Nuclear Data Center collaboration server at [nd-clx4.bnl.gov/gf/project/gnd](http://nd-clx4.bnl.gov/gf/project/gnd). Readers interested in translating an ENDF file into GND or making use of the other associated tools should download FUDGE and refer to the

documentation that has been provided along with the software for instructions on getting started.

### A. Resonance Reconstruction

In both ENDF and GND, resonance parameters can be used as a compact way to represent the rapidly fluctuating cross section at low incident energies. Both formats require the ability to reconstruct these resonance parameters in order to obtain a pointwise cross section. For ENDF files, the codes LINEAR and RECENT [11] handle resonance reconstruction, and for GND the code ‘fudgeReconstructResonances’ fills the same need. fudgeReconstructResonances is written mainly in Python (some numerically-intensive portions are written in C for better performance). It handles reconstructing both resolved and unresolved resonance parameters, including nearly all formats currently supported in ENDF (Single- and Multi-level Breit Wigner, Reich-Moore and the R-Matrix Limited formulae are all supported. The Adler-Adler formalism is not currently supported, since it appears to be unused in modern libraries). After reconstructing resonances, the code adds the resonance contribution to the background cross section, to obtain a pointwise cross section spanning both resonance and fast regions. One advantage of GND over ENDF is that the new pointwise reconstructed cross section is stored as a new cross section form, without overwriting the original

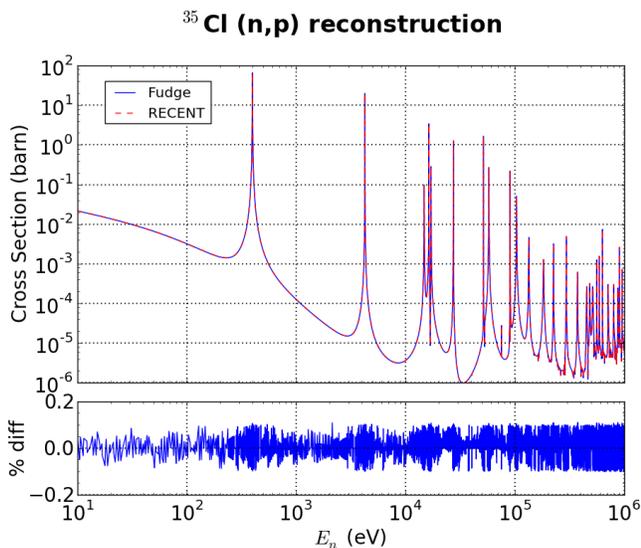


FIG. 3: Plot of the  $^{35}\text{Cl}$  (n,p) cross section, reconstructed from resonance parameters by the FUDGE code ‘fudgeReconstructResonances’ using the R-Matrix limited formalism. The cross section reconstructed by the code RECENT is also shown, as well as the percent difference between the two. While differences arise from the two codes using different sets of incident energies, the results agree to within the calculated tolerance of 0.1%.

data inputted by the evaluator.

An example of a cross section reconstructed from resonance parameters using fudgeReconstructResonances is shown in Fig. 3. The resonance parameters for this evaluation use the R-Matrix limited formalism (MF 2, LRF 7 in ENDF). For comparison, the result of reconstructing using the 2010 version of RECENT is also shown. In both codes, the cross section was reconstructed to a ‘tolerance’ of 0.001, meaning that linear interpolation anywhere in the resonance region should agree to 0.1% or better with the analytic value. As seen from the relative difference on Fig. 3, the two codes agree to within 0.1% at all energies.

### B. ‘toPointwiseLinear’

Since GND supports many different options for storing data, including piecewise functions and parameterized forms, a simple way for converting all these different forms into pointwise, linearly-interpolable data is needed. In FUDGE, the ‘toPointwiseLinear’ methods fill this need. All data types implement toPointwiseLinear except for the Madland-Nix energy spectra. If the original data were already in pointwise linear form, toPointwiseLinear returns a copy so that the user can make modifications without impacting the original. After conversion to pointwise linear form, data can be used for calculations or for plotting. The example in Fig. 4 shows a Kalbach-Mann double-differential distribution in the center-of-mass frame for neutrons from the reaction  $^{11}\text{B}(n,2n)^{10}\text{B}$  after conversion to pointwise linear.

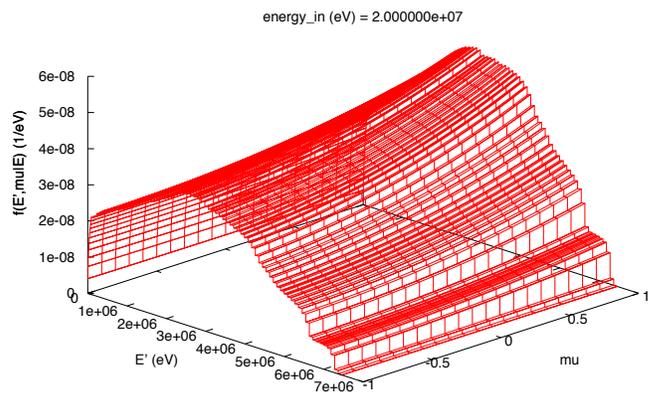


FIG. 4: Energy-angular distribution for outgoing neutrons from the reaction ‘n + B11 → 2n + B10’, at incident neutron energy 20 MeV. The distribution ‘nativeData’ for this reaction is Kalbach-Mann [12]. The distribution was converted to pointwise form using the toPointwiseLinear method.

### C. Checking codes

Quality assurance is one of the critical requirements when designing a structure for nuclear data. Quality as-

urance includes testing files to ensure they are properly formatted, and also checking that they contain consistent, physically realistic nuclear data. For ENDF-6 formatted files, quality assurance has depended to a large extent on the ENDF Utility Codes [13], including CHECKR for format checking, LINEAR for converting all interpolations to linear, and FIZCON and PSYCHE for testing the physics content. In the next few sections we discuss codes intended to provide the same types of quality assurance for GND.

The tools discussed in this section are specific to particular implementations of GND. Most of these tools depend on the FUDGE software package, or are specific to GND/XML.

### 1. XML Schema

An important requirement for a nuclear data format is a tool for automatically checking that files correctly follow the format. For ENDF-6 formatted files, the code ‘CHECKR’ was designed to fill this need. CHECKR is used to test ENDF files to ensure they follow the ENDF-6 format, and to give informative messages if format errors are encountered. The code parses an entire ENDF-6 file, checking for format-specific problems such as illegal values, missing sections, and disagreements between integer flags and the datasets they describe.

Checking GND/XML formatted data is made relatively simple by leveraging an existing technology called the ‘XML schema language’ [14]. This language is one of many tools that have been created for use with XML files, due to the popularity of XML as a format for storing data. An XML schema for GND has been created, and serves to define the hierarchy that must be followed in order for a GND/XML file to be ‘correct’. For example, the schema indicates that a reactionSuite must first contain a styles element, followed by documentations, particles, and so on (following the layout described in Sec. II). For each of those elements in turn, the schema defines which attributes, subelements and datasets are required, and which are optional.

The XML schema for GND/XML is distributed along with FUDGE. The schema (found in ‘fudge/gnd/gnd.xsd’) can be used on UNIX systems with the built-in command-line tool ‘xmllint’ (‘XML line interpreter’) to check a GND/XML file for format errors. For example:

```
xmllint --schema gnd.xsd GND_file.xml
```

If format errors are discovered in the file, xmllint will print an informative message, including line and column numbers, for each error. Once a file passes this step without errors, it has been ‘validated’ by the schema and can be considered a properly-formatted GND/XML file. This automatic format checking is a valuable tool for ensuring the quality of a nuclear data library.

### 2. Checking physics content

In addition to checking for format errors in GND, we also require tools to help ensure that the file contains physically realistic data. For ENDF files, this physics checking has been performed by the combination of two codes, FIZCON and PSYCHE. Together, they check for problems such as unnormalized probability distributions, negative cross sections or multiplicities, non-conservation of energy, incomplete quantities (not covering the full energy span required for an evaluation), and so on. Errors uncovered by FIZCON and PSYCHE are reported along with the section (identified by MF and MT numbers) where they occur.

Physics checking is also integrated into the FUDGE implementation of GND. In this case, physics checking is implemented with ‘check’ methods that are defined at several different levels of the Python classes representing the GND hierarchy. When the user checks a reactionSuite element, the check method first reconstructs resonances, calculates average energy per product, and then proceeds to call the check method of each reaction element within the reactionSuite. The reaction elements check their cross sections and their list of outgoing products including distributions and multiplicities. Last, each reaction is checked for energy balance. That is, whether the sum of the average kinetic energy to the products is equal to the available energy (Q-value plus kinetic energy of the projectile) for the reaction.

Using FUDGE to check GND/XML formatted files requires only three lines in Python:

```
>>> from fudge.gnd import reactionSuite
>>> r = reactionSuite.readXML("gnd_file.xml")
>>> print( r.check() )
```

An example of the types of warnings produced from the check methods is shown below for  $^{232}\text{Th}$  of the ENDF/B-VII.1 evaluation. In this example, the  $^{232}\text{Th}$  ENDF evaluation was first converted into GND and then the check method was called and returned the following warnings:

```
>>> print( r.check() )
ReactionSuite: n + Th232
  Reaction label 2: n + Th232_e2
    Product: n
      Distribution angular - LegendrePointwise:
        WARNING: Domain does not match the
        cross section domain:
    ...
  Reaction label 44: n + H1 + Ac231 + gamma
    WARNING: Calculated and tabulated Q-values
    disagree: -7162308.98 eV vs -7.755e6 eV!
    Product: H1
      Distribution Legendre - LegendrePointwise:
        WARNING: Unnormalized distribution!
        At energy_in = 2.1e7 eV (index 20),
        integral = 1.00001190074
    ...
```

TABLE XVI: Partial list of the tests performed by FUDGE when using the reactionSuite and covarianceSuite physics ‘check’ methods, along with the GND element where each test is implemented.

Test	GND element
Are any errors encountered during resonance reconstruction?	reactionSuite
For each complete reaction (see Sec. VE), do Z and A balance?	reaction
Does the Q-value agree with value computed from particle masses?	reaction
Do errors appear when computing average energy/momentum for products?	reaction
Does energy balance (sum of energy deposited to products equal to incident energy plus Q)?	reaction
Is the cross section threshold consistent with the Q-value?	crossSection
Does the cross section span the interval from threshold up to the maximum (default maximum=20 MeV)?	crossSection
Can the cross section nativeData form be converted to pointwise, linearly interpolable form?	crossSection
Does the cross section contain any negative values?	crossSection
Do all ‘transportable’ particles have distribution information?	product
Are any product multiplicities negative?	multiplicity
Does multiplicity domain agree with cross section domain?	multiplicity
Does distribution domain agree with cross section domain?	distribution
Do distribution frames make sense (lab vs. center of mass)?	distribution
Are distributions normalized?	distribution
Are any probabilities less than 0?	distribution
Does the matrix have any negative eigenvalues? Is the variance positive?	covarianceMatrix
Does the matrix have very small eigenvalues?	covarianceMatrix
Are variances (diagonal of the matrix) unreasonably large or small?	covarianceMatrix

The first warning indicates that for the reaction with output channel "n + Th232\_e2" (ENDF MT 52), the cross section is given on a different domain than the distribution for the outgoing neutron. The other two warnings are for the reaction with output channel "n + H1 + Ac231 + gamma" (ENDF MT 28) where there is a discrepancy between the listed Q-value and the Q-value calculated from particle masses, and an unnormalized probability distribution for the proton.

In addition to the reactionSuite check code, FUDGE also includes an set of checks that apply to the covarianceSuite. These are implemented in the same fashion as the reactionSuite checks (that is, ‘check’ methods are defined for several of the Python classes making up the covarianceSuite hierarchy, and are called in a nested hierarchy until all sections in the covarianceSuite have been checked).

Many possible warnings may be raised by the reactionSuite and covarianceSuite checking codes. They are listed in Table XVI. Many of the tests listed in this table can be modified based on the needs of the user. For example, by default FUDGE raises a warning if the cross section does not extend up to 20 MeV, but this limit can be increased or decreased by adding extra options when calling the check method. Also, the tolerances used when checking probability distribution normalizations, checking energy balance, and checking for agreement between Q-value and cross section thresholds can all be modified by user-supplied options.

In addition to checking, FUDGE has some capabilities for automatically fixing data where possible. For example, probability distributions can be renormalized. Unfortunately, many of the problems that occur in evalu-

ated nuclear data are not amenable to automatic fixing, instead requiring some manual intervention by the evaluator.

The checking methods are expected to play a central role in quality assurance for future evaluated nuclear data libraries, and have already contributed to finding many issues in the ENDF/B-VII.1 library before its final release.

#### D. Plotting Tools

FUDGE provides two separate plotting packages. The first uses Gnuplot and the Python wrapper package Gnuplot.py to provide in-line plotting for all FUDGE ‘XYs’, ‘W\_XYs’ and ‘V\_W\_XYs’ data containers. This plotter is invoked by calling the ‘plot’ method of any ‘XYs’, ‘W\_XYs’ and ‘V\_W\_XYs’ data container. For example, in an interactive Python session the following commands plot the total cross section from the file ‘zr90.endf’:

```
>>> from fudge.legacy.converting \
...     import endfFileToGND as endf2gnd
>>> r,c = endf2gnd.endfFileToGND( "zr90.endf" )
>>> r.reconstructResonances()
>>> r.getReaction("total").crossSection \
...     .toPointwiseLinear().plot()
```

A screen snapshot of this plot is shown in Fig. 5.

FUDGE also provides a Matplotlib-based plotting interface. To use this plotting interface the Python packages Numpy and Matplotlib have to be installed.

A standalone script called pltUtil.py has been provided in the ‘examples’ directory of FUDGE, which plots cross

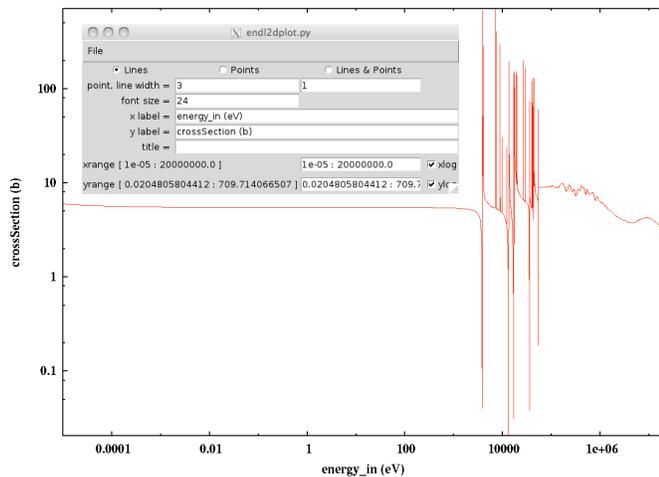


FIG. 5: Screen snapshot of a typical interactive plot window from the Gnuplot based in-line ‘plot’ method showing the ENDF/B-VII.1  $^{90}\text{Zr}(n,\text{tot})$  cross section.

sections from ENDF-6 formatted files by converting the files into GND within python. This script, when coupled with the x4i package [15], can plot experimental data from the EXFOR database [16] along side the evaluated data in an ENDF file. The script automatically reconstructs resonances and converts covariances into uncertainty bands. It can also interpret lumped reaction covariances to provide summed cross sections and uncertainty bands. Fig. 6 shows a screen snapshot from pltUtil.py used to view the  $^{241}\text{Am}(n,2n)^{240}\text{Am}$  cross section. In this figure, the evaluated data are given as a linear spline, hence it looks smooth. The covariance is given relative to values of the spline function at the linear spline nodes, but the covariance itself is grouped, with group boundaries aligning with the spline nodes. Therefore, the uncertainty jumps as one goes from one group to the next.

## VIII. TRANSLATING ENDF-6 FORMATTED FILES TO AND FROM GND

One of the major goals of the GND project has been to provide backwards compatibility with the ENDF-6 format. This is motivated partly by the fact that, since many existing codes and programs that use ENDF-formatted data are currently in use for important applications world-wide, users will likely adopt a slow and cautious approach to moving away from ENDF. GND is expected to co-exist alongside the ENDF format for many years, during which the ability to translate back and forth will be an important way for GND to start being actively used and to gain acceptance by the broader nuclear data community. In this section, we describe the current status of the translation tools, which we hope will facilitate the transition to GND.

The FUDGE software package includes a function called

‘endfFileToGND’, that reads an ENDF-6 formatted file, and translates it into a Python representation of the GND structure. Users may prefer to use the FUDGE tool ‘rePrint.py’, found in the ‘bin’ directory of a FUDGE distribution. This tool calls endfFileToGND to translate an ENDF file, and outputs the results to an XML file. In addition, rePrint.py translates from GND back into ENDF (hence the name ‘rePrint’) so that the data can easily be compared with the original file to discover what was lost or modified during this two-way translation.

The code ‘rePrint.py’ produces several different output files. The most important are ‘test.endf6.xml’ (containing the new reactionSuite in GND/XML format), ‘test.endf6-covar.xml’ (containing the covarianceSuite in GND/XML), and ‘test.endf6.noLineNumbers’ (an ENDF-formatted file produced by translating GND back into ENDF-6 format). The original ENDF data is also rewritten with line numbers removed for easier comparison. The original and re-printed ENDF-6 files can be compared using tools like ‘diff’ or ‘kompare’ in order to discover what differences were produced during translation. Readers who are interested in testing out GND are encouraged to start by translating ENDF files to see what is preserved or changed during the translation, and to compare how data are represented in ENDF and in GND.

An ENDF-6 formatted file can be translated into GND and then retranslated back to ENDF. The retranslated ENDF-6 formatted file is nearly identical to the original. Some important differences remain, however, often related to the use of less common ENDF-6 format options. Since some of these options are not yet supported by GND, these types of data will be missing from the retranslated ENDF-6 file. Examples include covariance matrices for the unresolved resonance region (MF 32, LRU 2 in ENDF) and production of beta-delayed gammas following fission (MF 1, MT 460 in ENDF). These data types will be handled in the future as time and manpower permit. Some other differences, however, arise due to design choices that were made in implementing GND. These differences will not disappear in the future. Some of the most common differences that appear after translating back and forth between ENDF and GND are discussed below.

### A. Converting pointwise to piecewise

One difference that often shows up after translating to and from GND is the introduction of additional piecewise regions. This change is due to a design choice made when implementing GND. In GND, the independent axis of a pointwise function is not allowed to contain the same point more than once. This choice was made in order to prevent users from having to deal with multiple-valued functions. If evaluators wish to use a step function in GND, they must either implement it at the boundary between two piecewise regions, or provide two points in

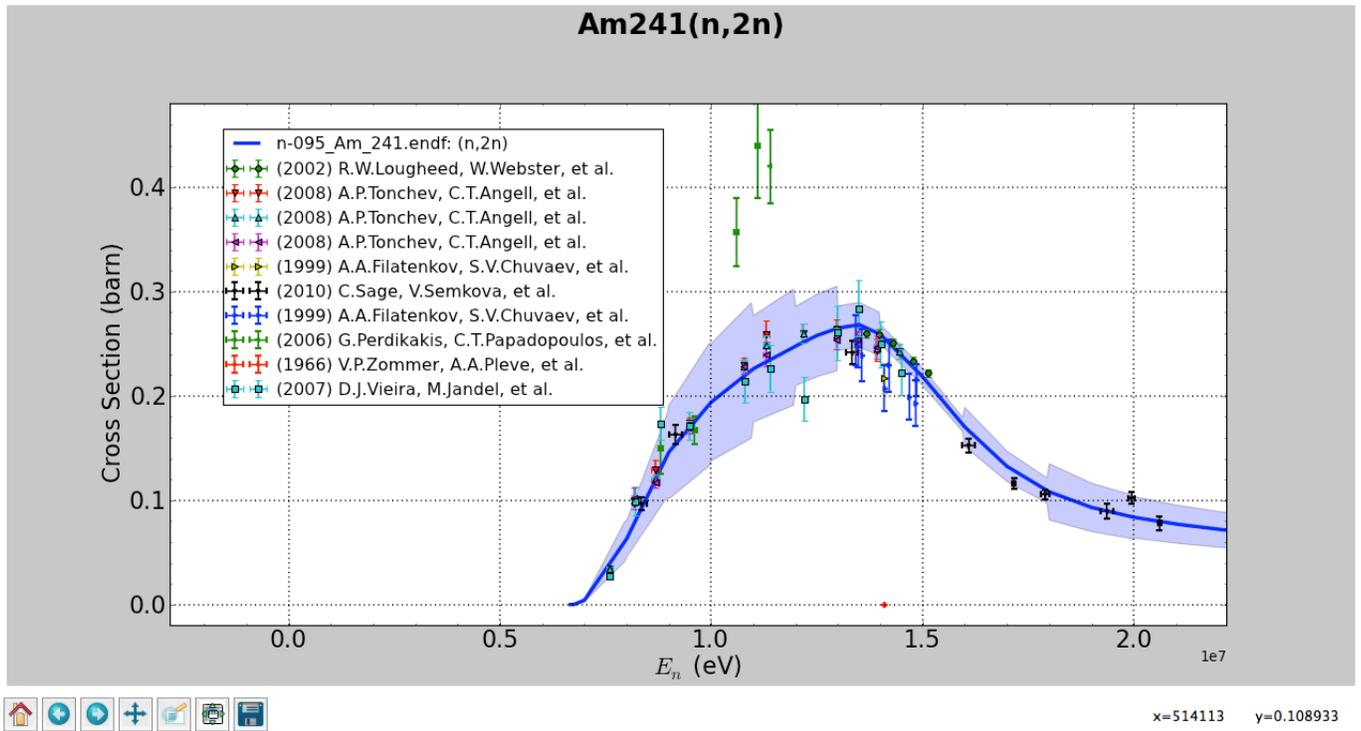


FIG. 6: Screen snapshot of a typical plot window from the Matplotlib based `pltUtil.py` script included in the FUDGE package. This script, when coupled with the `x4i` package, can plot EXFOR data along side the evaluated data in an ENDF file. This figure shows the  $^{241}\text{Am}(n,2n)$  cross section from ENDF/B-VII.1.

a pointwise region with a small epsilon between the x-values.

This design choice is more restrictive than ENDF-6, in which two or more points with the same x-value sometimes appear. For example, identical x-values may appear in energy-dependent cross section or multiplicity data. As a result, when these files are translated to GND and back, the resulting ENDF-6 formatted file will differ from the original, containing two or more piecewise regions in what was originally a single region. An example of this type of difference is seen in Table XVII. The reader should note that the new data still uses the same interpolation as the original, it is simply broken up into two piecewise regions. This change will not have an impact when using the data in transport codes as long as the ENDF processing codes are properly interpreting the ENDF format.

## B. Particle masses

Another common difference between the original and re-printed ENDF files is in particle masses. This difference arises since the ENDF-6 format stores the masses of the target, projectile and reaction products (given as multiples of the mass of the neutron) many times throughout the file. Frequently, these values are inconsistent, sometimes varying between different MF sections, and some-

TABLE XVII: Sample ENDF-formatted interpolation data before and after translating from GND (this example is taken from the ENDF/B-VII.1 neutron sub-library, MAT 2525, MF 3, MT 2). Before translation, all 84 points are in the same interpolation region, but afterwards they are split up into two separate regions. In the re-printed ENDF file, only the interpolation information changes; the data are unaffected.

Original ENDF file:					
2.505500+4	5.446610+1	0	0	0	0
0.000000+0	0.000000+0	0	0	1	84
84	2	0	0	0	0
1.000000-5	0.000000+0	1.250000+5	0.000000+0	1.250000+5	5.729550+0
After translation:					
2.505500+4	5.446610+1	0	0	0	0
0.000000+0	0.000000+0	0	0	2	84
2	2	84	2	0	0
1.000000-5	0.000000+0	1.250000+5	0.000000+0	1.250000+5	5.729550+0

times changing even within the same section. Based on discussions with many ENDF evaluators, we have come to the conclusion that these discrepancies likely appeared gradually, as different sections of ENDF files were updated but not kept consistent with other sections. This has unfortunately led to a situation where the original value chosen by the evaluator is no longer clear.

In GND, we propose eliminating this type of uncertainty by only storing the mass of each particle once in the file, either in the particles element at the start of the file, or in the future in an external particle database like that described in Sec. IX F. This change is already being implemented by the `endfFileToGND` translation code: if an ENDF file contains multiple masses for a particle, the translator chooses the value that appears most often in the file. As a result, the discrepant particle masses will be overwritten in the reprinted ENDF file. This change means that evaluators seeking to update an old evaluation should either use the same values used in the original version, or commit to updating the mass,  $Q$ -values and thresholds throughout the entire evaluation.

### C. Handling ENDF-6 format errors

Some ENDF files do not follow the ENDF-6 format as described in the current manual [1]. These files cause problems both for end users, and for translation codes like `endfFileToGND`. Some common format errors that have been encountered while developing the ENDF translator include: energy-dependent multiplicities given for reactions like  $(n,2n)$  where the multiplicity should be constant, `Inf` or `NaN` appearing in ENDF files, and many cases where incident energies are not correctly sorted in ascending order. Other common problems are listed in Table XVIII. The `endfFileToGND` translator will process the ENDF file, skipping any sections which it detects as having format issues, as it is beyond the scope of a translation code to fix ENDF-6 format errors. After translating the ENDF file, `endfFileToGND` will print all errors detected. The translator will then either abort (by raising a Python ‘Exception’) or will return a GND structure with the bad data sections missing, depending on value of the ‘`skipBadData`’ option. When using ‘`rePrint.py`’, the ‘`skipBadData`’ option can be turned on by adding the ‘`--skipBadData`’ as

```
python rePrint.py --skipBadData file.endf
```

With this option, the translator will return a GND `reactionSuite` even if errors are encountered during translation. The reader should note, however, that when using this option, format errors usually result in complete sections of ENDF data being skipped by the translation code, hence those sections will be missing from the final GND file and the re-printed ENDF file. As much as possible, the warnings and errors encountered during translation of ENDF files should be corrected in the original ENDF file in order to ensure the final GND file is correct.

Since ‘`endfFileToGND`’ attempts to strictly adhere to the ENDF-6 format as stated in the format manual, it played a significant role in cleaning up issues prior to the release of the ENDF/B-VII.1 library [17]. Each beta-release prior to the official VII.1 release was tested by converting to GND. All format errors discovered during

this process were reported back to evaluators and to the library maintainers. As a result, most of these format errors (including those seen in Table XVIII) were fixed in time for the official release of ENDF/B-VII.1.

## IX. FUTURE WORK

Version 1.1 of the GND structure is already available to the nuclear data community for testing. More work remains, however, before GND can begin to be adopted by the community as a replacement for older formats. In this section, some of the ongoing work towards improving GND is described. This work includes further additions and revisions to the structure of GND, tools for processing the data into forms suitable for use in transport simulation codes, access routines that will allow users to efficiently use GND in their codes, and also possible future projects that could build on the success of GND in handling evaluated nuclear reaction data, by changing how experimental data and nuclear structure information are stored.

### A. Finalizing the Structure of GND

The current versions of FUDGE and GND represent an effort by LLNL to propose a new standard way to store nuclear reaction data. GND now needs feedback and criticism from the international community in order to ensure that it meets the needs of users. A new WPEC subgroup (number 38) has been formed with the goal of modernizing the storage of nuclear data, and this subgroup is expected to hasten the process of collecting user feedback and reviews, and hasten the transition toward a new international standard [18]. Based on this engagement with the broader community, changes to the GND structure will likely be made before this transition is complete. Once both CSEWG and the international community are satisfied with the GND structure and tools, ENDF libraries will begin to be officially released in GND. CSEWG anticipates many years in which ENDF libraries will be released in both the GND and the legacy ENDF-6 formats (this process is possible since translating between the formats is straightforward using FUDGE). This period of parallel releases of the ENDF library will allow other labs to port their processing codes (including NJOY [5] from Los Alamos and AMPX from Oak Ridge [19]) and other tools (including websites like those hosted by the National Nuclear Data Center) to use GND.

GND must also be extended to handle more of the evaluated nuclear reaction data available in major libraries. Development of GND originally focused on neutron-induced reactions, and has since been extended to also handle incident gammas and charged-particles. As seen in Table XIX, several other sub-libraries are also available and should be addressed by GND.

TABLE XVIII: Some common format problems encountered when translating ENDF-6 files into GND.

---



---

-Incident energies not sorted in ascending order, usually for Legendre coefficients (MF 4, LTT 1) or Kalbach-Mann systematics (MF 6, LAW 1, LANG 2)
-MF 2 and MF 32 both contain a list of resonance parameters. These are redundant and should always be identical, but sometimes they contain discrepancies.
-Mismatch between excited state energies listed in different parts of the file (MF 4 versus MF 12, for example).
-Product disagrees with MT. For example, $^{186}\text{W}(n,t 2\alpha)$ should produce $^{176}\text{Tm}$ , but is listed as producing $^{176}\text{Yb}$ .
-Disagreement between the number of gammas listed in MF 12 or MF 13 and the number listed in MF 14.
-Covariance matrices using the sparse (LCOMP 2) format, attempting to assign to an out-of-bounds matrix element.

---



---

TABLE XIX: Status of the GND implementation of features needed in each ENDF/B-VII sublibrary

No.	NSUB	Sublibrary	Status
1	0	Photonuclear	Fully implemented
2	3	Photo-atomic	Not implemented
3	4	Radioactive decay	Needs final structure format
4	5	Spontaneous fission yields	Not implemented
5	6	Atomic relaxation	Not implemented
6	10	Neutron	Fully implemented
7	11	Neutron fission yields	Not implemented
8	12	Thermal scattering	Not implemented
9	19	Standards	Fully implemented
10	113	Electro-atomic	Not implemented
11	10010	Proton	Fully implemented
12	10020	Deuteron	Fully implemented
13	10030	Triton	Fully implemented
14	20030	$^3\text{He}$	Fully implemented
N/A	N/A	Alpha	Fully implemented

### B. Processing codes for GND

As the structure of evaluated data in GND becomes finalized, the next important step is to enable processing the data for use in transport calculations. One important advantage of GND is that processed data (i.e., data suitable for Monte Carlo or deterministic transport codes) can be stored in the same hierarchy as the original evaluated data.

The FUDGE infrastructure can currently handle most of the processing of a GND file. For example, FUDGE contains methods for converting GND evaluated data into grouped data suitable for deterministic transport, and for heating the cross section data to a desired temperature. FUDGE also supports processing parameterized forms of data, such as Kalbach/Mann and Watt distributions, into transfer matrices suitable for deterministic transport. With the exception of Coulomb Scattering (ENDF data MF 6, LAW 5) which has a singularity issue, all ENDF energy-angle distribution forms can be

processed to grouped, deterministic transfer matrices.

The transfer matrices resulting from converting ENDF files to GND and then processing with FUDGE have begun to be compared to the equivalent matrices produced by processing with NJOY. For example, Fig. 7 shows the absolute and relative differences between the  $^{235}\text{U}$   $L=0$  fission transfer matrices produced by FUDGE and by NJOY. The largest absolute disagreement appears at small incident energy, where NJOY assumes a constant outgoing spectra for several incident energy groups. The largest relative difference between the two matrices is still less than 0.1%, however. This example is a fairly simple case, since the original neutron distributions were given in pointwise form. Transfer matrices produced from parameterized distribution forms do show some substantial differences when compared to NJOY, indicating that the FUDGE processing of those parameterized forms still requires some work.

The FUDGE package also contains methods for calculating the average energy (similar to the KERMA data in ACE files) and momentum of a product. These data are used in deterministic and expected-value Monte Carlo transport simulations.

### C. Transport code access routines (GIDI)

Once GND data can be processed, optimized access routines are also needed in order for transport codes to start taking advantage of the new structure. While the FUDGE package provides a high-level interface for reading, writing, viewing, checking and manipulating GND files, it is mainly written in the Python programming language which can be significantly slower than low-level languages like C. The Python code is much easier to write and smaller, but interfacing with transport codes will require access routines written in a low-level language for speed and memory reasons. A new application programming interface, or API, called GIDI (General Interaction Data Interface) is being developed to read a GND file. The core of GIDI will be written in C, and wrappers for other languages like C++ will also be provided in the API.

It is anticipated that in the near term GND files will only exist in the meta-languages XML and HDF5, and both will be supported by GIDI.

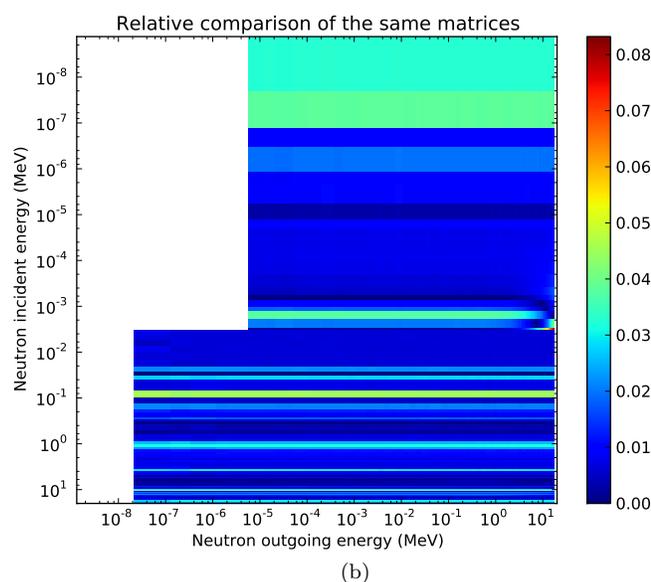
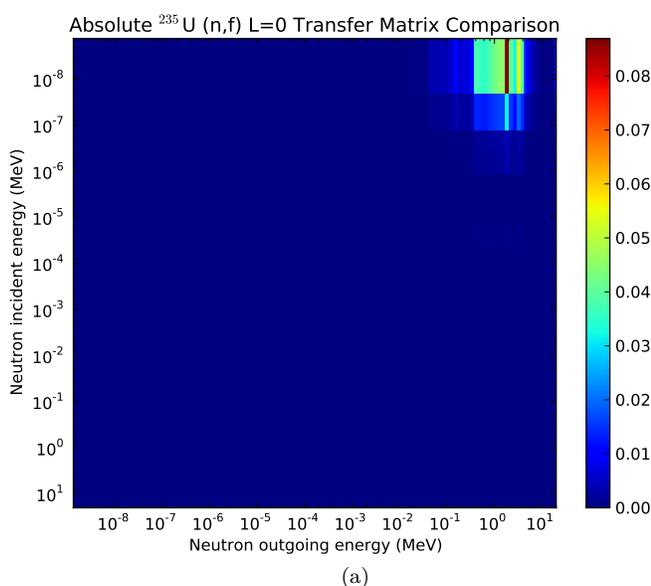


FIG. 7: Absolute (a) and relative (b) differences (in %) between the  $^{235}\text{U}$  (n,fission) transfer matrices produced by processing with FUDGE and with NJOY. Spurious differences appear in white at small outgoing energies in (b) since NJOY rounds these matrix elements down to zero. At all other incident and outgoing energies the codes agree to 0.1% or better.

#### D. Uncertainty quantification tools: Monte Carlo and deterministic uncertainty propagation

LLNL developed a tool called kiwi [20] that performs Monte Carlo uncertainty propagation using nuclear covariance data. This tool samples directly from the covariances in an evaluation to produce realizations of evaluations consistent with the underlying covariance. This tool was used successfully in a variety of uncertainty quan-

tification studies at LLNL. Unfortunately, Monte Carlo uncertainty propagation is time and computing resource intensive and can only practically vary tens of parameters simultaneously. Typical actinide covariances contain hundreds of rows and we can only effectively vary a small subset of the data.

The sensitivity matrix approach, in which one linearizes the response of a system to variations in nuclear data, allows one to propagate much more data in a uncertainty quantification study. This approach is very successful for problems that can be linearized, but can result in unrealistic answers when a system has a strong non-linear response to changes in the nuclear data. The nuclear data community is seeking to develop hybrid approaches to uncertainty quantification that marries the two approaches. Once this effort is sufficiently well developed, we will seek to port both the legacy kiwi system as well as newly developed uncertainty propagation tools to FUDGE.

#### E. Evaluator tools: creating new GND evaluations directly from Nuclear Reaction Modeling Codes

LLNL and BNL are both developing tools to translate the output of the popular Nuclear Reaction modeling codes TALYS and EMPIRE into GND format. Both projects anticipate integration of their scripts into the FUDGE code base sometime in FY13-FY14.

#### F. External particle database

As mentioned in Sec. VIII B, GND unifies all particle information (nuclear masses, levels, and gamma decays) into a single element, eliminating repetitious data. However, since each reactionSuite still stores its own copy of particle information, there is still a possibility for discrepancies between particle information in different evaluations. In the future, libraries could be made more uniform by allowing GND files to refer to an external “Properties of Particles” (POP) database.

POP is designed to allow storing all information for every particle involved in an evaluated nuclear reaction library in a single place, so that each evaluation can link to the same data. Information provided in POP will include properties of nuclei, gammas and electrons (along with other particles such as pions, as evaluations extend to higher energies).

A version of POP has already been implemented by adopting masses from the Atomic Mass Evaluation version 2003 (AME2003) [21], and level schemes and gamma decays from the Reference Input Parameter Library (RIPL-3) [22]. The data are stored in a hierarchy similar to the structure seen in Sec. VIII B. An example of how POP stores nuclear masses and energy levels is shown in Table XX.

The naming of elements and attributes in POP are cho-

TABLE XX: This table describes the ‘nucleus’ element from the POP database. For each nucleus (identified by atomic symbol and mass number) the mass element includes all the information provided in AME2003, and the nuclearLevels element contains level schemes taken from the RIPL-3 library.

---



---

- nucleus SymbolA
  - mass
    - \* atomic
    - \* bindingEnergyPerA
    - \* betaDecayEnergy
    - \* ...
  - nuclearLevels
    - \* nuclearLevel
      - energy
      - half-life
      - spin
      - parity
      - decay, particles such as  $\gamma$ ,  $\alpha$ ,  $\beta$ , etc.
      - ...
    - \* nuclearLevel
    - \* ...

---



---

sen to make the information accessible for users. Moreover, this way of organizing the data should allow one to easily select any information needed for one nucleus and easily integrate it with GND. GND allows evaluators to use an external database as well as to overwrite individual structure information.

While the POP database is being designed with evaluated nuclear reaction data in mind, it is also envisioned as a way to store evaluated nuclear structure data. If GND requires information beyond what is available in AME2003 and RIPL-3, then POP will be extended to provide that information, which may come from other evaluated libraries.

In the future, POP may also need to be extended by including other properties of particles, such as atomic or molecular properties, perhaps by taking advantage of previous work defining an atomic and molecular database [23]. Other properties of particles can be integrated by adding more elements similar to the nucleus element.

## X. CONCLUSION

Nuclear data was one of the earliest fields to take advantage of the digital computing revolution in the 1950s. The field has had a long and fruitful history, spurred along in part by the success and wide-spread adoption of the ENDF format as a standard way of sharing and using nuclear data. In the past 50 years, however, computing technology has seen tremendous growth, while nuclear data storage has remained relatively unchanged. The

Generalized Nuclear Data (GND) structure has been designed to change this by providing the nuclear data community with a modern structure that takes advantage of the improved technologies available today.

For both evaluators and users of nuclear data, GND offers several advantages over earlier formats. Some of the most important advantages include:

- arbitrary precision for numeric data (compared to the 7 digits of precision available using standardized ENDF format),
- removing the artificial limitations that have been imposed by the ENDF MT numbers on the number of discrete reactions allowed in an evaluation,
- support for separating reaction processes, better representing the physics behind the data and aiding the manipulation of the data by the evaluator,
- removing redundancies (and the potential for discrepancies) in evaluated data by using names and links,
- support for storing the evaluated data along with multiple different forms of derived data, so that the same tool set can be used to handle both evaluated and processed nuclear reaction data,
- easy extensibility, so that new types of data can be added with little effort, and
- a physics-based hierarchy that is easy for a novice to understand and navigate.

The FUDGE infrastructure also provides backwards-compatibility with ENDF-6, allowing users to translate data back to ENDF-6 and continue using codes written for ENDF-6 even after the transition to GND begins.

Initial steps towards adopting GND are already underway. In particular, the latest version of the ENDF library (ENDF/B-VII.1) is already available in GND format, and the US nuclear data community expects that future versions of the library will be released in both the ENDF-6 format and GND.

GND is not yet a finished product: the structure is expected to continue evolving in order to meet the needs of the nuclear data community. In particular, the new Working Party on Evaluation Cooperation (WPEC) subgroup 38 will play an important role in defining the new structure, and identifying the kinds of nuclear data needs that are still unmet and should be addressed by GND. The authors of this paper wish to invite any and all comments from the wider nuclear data community during this process, to help ensure that GND is a flexible and powerful tool for handling current and future nuclear data needs.

### Acknowledgments

We wish to acknowledge valuable review and feedback by M. Chadwick, M. Herman and P. Obložinský. This work was performed under the auspices of Department of Energy contract Nos. DE-AC52-07NA27344

(Lawrence Livermore National Laboratory) and DE-AC02-98CH10886 (Brookhaven National Laboratory). The project was partly funded through the Nuclear Data Program Initiative of the American Recovery and Reinvestment Act (ARRA).

- 
- [1] “ENDF-6 Formats Manual: Data Formats and Procedures for the Evaluated Nuclear Data Files ENDF/B-VI and ENDF/B-VII”, ed. A. Trkov and D. Brown, BNL Report BNL-90365-2009 Rev.2, CSEWG Document ENDF-102 (2011). See: <http://www.nndc.bnl.gov/csewg/docs/endl-manual.pdf>
- [2] The Extensible Markup Language (XML) 1.0 (Fifth Edition). See: <http://www.w3.org/TR/REC-xml/>.
- [3] The HDF Group. Hierarchical data format version 5, 2000-2010. See: <http://www.hdfgroup.org/HDF5>.
- [4] B.R. Beck, “FUDGE: A Program for Performing Nuclear Data Testing and Sensitivity Studies”, AIP Conf. Proc. **769**, 503 (2004).
- [5] R.E. MacFarlane, A.C. Kahler, “Methods for Processing ENDF/B-VII with NJOY”, Nuclear Data Sheets, **111**, Issue 12, 2739 (2010). See: <http://t2.lanl.gov/codes/njoy99/>
- [6] M. Herman, R. Capote, B.V. Carlson, *et al.*, “EMPIRE: Nuclear Reaction Model Code System for Data Evaluation”, Nuclear Data Sheets **108**, 2655 (2007).
- [7] A. Koning, A. Hilaire, S. Goriely, “TALYS-1.4 User’s Manual”, Nuclear Research and Consultancy Group (NRG), (2011). See: <http://www.talys.eu/fileadmin/talys/user/docs/talys1.4.pdf>
- [8] XML Linking Language (XLink) Version 1.1. See: <http://www.w3.org/TR/xlink11>
- [9] N. M. Larson, “Updated Users’ Guide For SAMMY: Multilevel R-Matrix Fits To Neutron Data Using Bayes’ Equations”, ORNL Report ORNL/TM-9179/R8, CSEWG Document ENDF-364/R2 (2008).
- [10] N. M. Larson, “SAMMY User Guidance for ENDF Formats”, ORNL Report ORNL/TM-2007/23, CSEWG Document ENDF-367 (2007).
- [11] D. Cullen, “PREPRO 2010: 2010 ENDF/B Pre-Processing Codes”, IAEA Report IAEA-NDS-39, Rev. 14 (2010).
- [12] C. Kalbach, “Systematics of Continuum Angular Distributions: Extensions to Higher Energies” Phys. Rev. C. **37**, 2350 (1988).
- [13] C.L. Dunford: ENDF Utility Codes Release 7.01/02, Released April 2005
- [14] C.E. Campbell *et al.*: “XML schema”, ACM SIGMOD Record *32* issue 2 (2003)
- [15] D. Brown, “x4i: the EXFOR interface v1.0” (2011). See: <https://ndclx4.bnl.gov/gf/project/x4i/>
- [16] V. McLane “EXFOR Basics” IAEA Report IAEA-NDS-206 (2000). See: <http://www.nndc.bnl.gov/exfor/>
- [17] M.B. Chadwick *et al.*, “ENDF/B-VII.1: Nuclear Data for Nuclear Science and Technology: Cross Sections, Covariances, Fission Product Yields and Decay Data”, Nuclear Data Sheets **112**, 2887 (2011).
- [18] Working Party on International Nuclear Data Evaluation Co-operation (WPEC). See: <http://www.oecd-nea.org/science/wpec/>.
- [19] N.M. Greene, “The AMPX-2000 Operating System for Producing Continuous Energy and Multi-Group Cross Sections from Basic Data Libraries Using the ENDF/B-6 Formats”, Proceedings of the International Conference on Nuclear Data for Science and Technology (ND2001), Oct. 7-12, 2001, Tsukuba, Japan.
- [20] C.M. Mattoon *et al.*, “Covariance Applications with Kiwi”, EPJ Web of Conferences **27** (2012).
- [21] G. Audi *et al.*, “The Ame2003 atomic mass evaluation (II)”, Nuclear Physics **A729**, 337 (2006).
- [22] R. Capote *et al.*, “Reference Input Parameter Library (RIPL-3)”, Nuclear Data Sheets **110**, 3107 (2009).
- [23] XSAMS: XML Schema for Atoms, Molecules and Solids. See: <http://www-amdis.iaea.org/xsams/about.html>