

Introduction

Lawrence Livermore National Laboratory (LLNL) is releasing version 1.1 of the new “Generalized Nuclear Data” (or GND) structure and framework, designed to replace the ENDF-6 and LLNL’s ENDL formats. In this package, we are releasing a set of routines (written in Python) that will convert an existing ENDF-6 formatted file into the new format. The software and format are still in beta form, so we are hoping for feedback from the nuclear data community on the design of the new format. That is, we need people to review our current format!

By default, this package outputs the new format in XML (for converting to HDF5, see the section 'XML to HDF5' below). However, it is important to understand that what we are developing is a structure for representing nuclear reactions and their data, and not the particular language (e.g., XML, HDF5) that will be used.

This document has seven sections: 1) **Introduction**; this section, 2) **Reaction Nomenclature**; contains several definitions used in this manual, 3) **Structure**; gives a brief description of the nuclear reaction format we are proposing, 4) **ENDF to GND and back to ENDF**; describes how to use the python routines in this package to convert an ENDF file into an internal python representation of the proposed format, write this representation to an XML file and rewrite it back into an ENDF file, 5) **Schema**; describes how to check the structure of an XML file against a set of rules for the new format, 6) **XML to HDF5**; describes converting an XML file to an HDF5 file, and 7) **Issues**; discusses some of the differences between the original ENDF file and the rewritten ENDF file produced by rePrint.py (as described in section **ENDF to GND and ENDF**).

Reaction Nomenclature

In this document and in the XML file, the name of a nuclear particle is labeled as “SA[e]” where “S” is the symbol for the particle (e.g., “O” for Oxygen), “A” is the particle’s nucleon number and “e” is an optional (as denoted by “[” and “]”) excitation or meta-stable level designator for the nucleus. Level designators are labeled as “eN” where “N” is the level number (e.g., “e5” for the fifth excitation level) and meta-stables are labeled as “mN” (e.g., “m2” for the second meta-stable). A particle may also be given a level designation of ‘continuum’ or ‘sum of all levels’ using “O16_c” and “O16_s” respectively. For a neutron, the symbol “n” is used and “N” is used for nitrogen. As example, neutron is “n”, Oxygen 16 in the ground and third excited states are “O16” and “O16_e3” respectively, and Americium 242 in the first meta-stable state is “Am242_m1”. Note that for neutrons and light ions, the nucleon number 'A' is no longer included: the symbol for a neutron is simply "n".

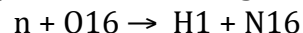
A product is a special kind of particle that has a multiplicity and an energy/angular spectrum. For example, the product “n[multiplicity=’3’]” may be used to describe the three neutrons produced in an (n,3n) reaction where each neutron has the same energy/angular spectrum. However, if the three neutrons each have a different spectrum, then one would represent this as three separate products (e.g., “n + n + n”, see nomenclature on reaction channels below). If no explicit multiplicity is given, a multiplicity of 1 is implied. A product’s multiplicity can be an integer (e.g., “He4[multiplicity=’2’]” for two Helium-4 particles) or it can be energy dependent (i.e., “n[multiplicity=’energyDependent’]”). Energy dependent multiplicities can occur for neutrons emitted during fission or for gammas, as example. If the multiplicity is energy dependent then the product must contain a “multiplicity” element.

A reaction channel is listed as

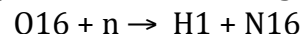


where body can be a single product, or a product and its breakup products. A product and its breakup products are enclosed by “(“ and “)” as “(product \rightarrow decayProduct1 + ... + decayProductN)”. Here are several examples:

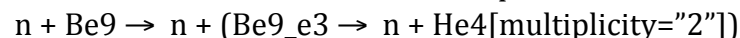
Example 1, an “n” hitting an “O16” producing a “H1” and “N16”,



Example 2, an “O16” hitting an “n” producing “H1” and “N16” (note the projectile must come first),

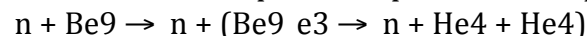


Example 3, a “n” hitting “Be9” producing a “n” and “Be9” in the third excited state with the “Be9” decaying into a “n” and two “He4”s where one distribution is used to describe both “He4” particles,



In this example, the XML file will contain two decay products (i.e., “n” and “He4”) under the “Be9_e3” product.

Example 4, same as example 3 except each “He4” particle has its own distribution data,



In this example, the XML file will contain three decay products (i.e., “n” and two “He4”s) under the “Be9_e3” product.

Structure

This section describes the general output of the proposed structure and does not go into details (see the sample GND files in the ‘examples’ directory for more detail). Furthermore, this section will describe things from the XML language perspective.

We are proposing a hierarchical structure for a single target being hit by a projectile. Each file contains the suite of all reactions possible for a target/projectile combination. The basic structure of the file is as follows:

reactionSuite: this specifies the target and projectile, and is the top-level element containing everything else. A target has a temperature, but is, otherwise, assumed to be at rest on average. In XML form, the top element of a “projectile + heated target” file is called a “reactionSuite”. This element contains the following children element objects:

styles: this element contains the styles that are represented in the data. Possible styles include evaluated, Monte Carlo and deterministic representation of nuclear data. This allows the reuse of this format to represent various types of nuclear data that all share the same general structure for representing reactions. In the XML files outputted by this package, the only style will be “evaluated”.

documentation: this is the documentation for this target and is similar to ENDF MF=1, MT = 451 data. The current plan is that the “reactionSuite” element will require a “documentation” element, and each “reaction” element may also optionally contain a “documentation” element. ENDF-generated documentation contains only plain ASCII text, however the documentation may be written in HTML.

particles: this element contains a list of all particles, and their properties, that are inputs or outputs of the reaction channels listed in the file.

resonances: this element contains the resonance energies and widths for both resolved and unresolved resonance regions. All formats available in ENDF (Single- and Multi-level Breit-Wigner, Reich-Moore and R-Matrix Limited) are supported in the new format.

reaction: this element represents a single reaction. There are often many of these elements, one for each reaction. This is similar to many of the ENDF MT numbers (e.g., 2, 50, 51, 600, 16, 18). A reaction contains a <crossSection> and an <outputChannel> (containing all information about the reaction products). The reaction elements should not overlap. That is, summing up the cross sections for all reaction elements should produce the total cross section.

-crossSection: contains the reaction cross section. Cross section data may be ‘pointwise’ (a single region with the same interpolation throughout), ‘piecewise’ (multiple regions, usually with different interpolations in each region), or ‘backgroundWithResonances’ (indicating that reconstructed resonance data must be added to the background data to achieve the full cross section).

-outputChannel: this element (found inside a ‘reaction’ element) contains the list of outgoing products. Each product will contain a multiplicity and, if known, spectral data (i.e., some form of the normalized differential cross section $d^2\sigma(E)/dE'/d\theta / \sigma(E)$ where $\sigma(E)$ is the integrated cross section, E is the projectile energy, and E' and θ are the outgoing energy and angle, respectively). The outputChannel also has a ‘genre’, which may be one of the following:

-twoBody: this outputChannel initially has only two outgoing products (e.g., $n + O^{16} \rightarrow H^1 + N^{16}$ or $n + Pu^{239} \rightarrow n + Pu^{239}_{e4}$). For a two-body channel, only the angular data for the outgoing products are required, as the outgoing energy can be calculated from the angle. One or both of the products may further breakup (e.g., $n + Be^9 \rightarrow n + (Be^9_{e4} \rightarrow n + He^4 + He^4)$).

-uncorrelated: this represents channels that are not two-body; hence, if known, both outgoing energy and angular data must be present. Furthermore, each outgoing particle is assumed to be uncorrelated to the other outgoing particles. Fission is also listed under “NBody”, however, we anticipate making a separate genre, (e.g., “fission”) for fission in the future.

-sumOfRemainingOutputChannels: this channel is equivalent to ENDF MT = 5 data. That is, it represents the sum of all channels not listed under the other ‘channel’ elements. Like an NBody channel, for each product both outgoing energy and angular data must be present. And, each outgoing particle is assumed to be uncorrelated to the other outgoing particles.

summedReaction, *production* and *fissionComponent*: as mentioned above, reaction cross sections in GND should always sum up to the total cross section. Often, however, evaluations contain information both about various discrete reactions and also about their sum. The most important example is the total cross section (MT 1), which is provided along with all its constituent reactions. In GND, this should be stored in a *summedReaction* element rather than a reaction element, to indicate that the data is in principle redundant. GND also provides a *production* element for storing the cross section for producing a radioactive product, and a *fissionComponent* element for storing first-chance, second-chance etc. fission cross sections.

Special note on 'xData' elements:

In GND, some elements contain an 'xData' attribute. These elements contain the actual data. They may be 1-dimensional (cross sections and multiplicities), 2-dimensional (outgoing angular or energy distributions), or 3-dimensional (double-differential distributions). xData-type elements always contain a list of axes (with label and units), followed by the data.

Sample GND files are available in the 'examples' directory in this release.

In addition to the 'reactionSuite', the GND format defines a 'covarianceSuite', written to a separate file from reactionSuite, that contains all covariance matrices for the evaluation. The covariance format in GND is still preliminary, but sample covarianceSuite files are available both in the 'examples' directory and by using the 'rePrint.py' utility as described below.

ENDF to XML and back to ENDF

Quick start:

To try out the new format, simply place the included package in the current directory, and do (from the command line):

```
>tar -xf GND-v1.0.tar.gz    # a new directory named "fudge-2.0" will be created
>cd fudge-2.0
>python bin/rePrintSample.py
```

This will translate a sample ENDF file into an internal python representation of the new format. It will then write the new format in XML format in files named 'test.endf6.xml' and 'test.endf6-covar.xml', and will also convert back to ENDF format and write the resulting file to 'test.endf6'. If the 'xdiff' tool is available on your system, you will automatically see a comparison between the original and re-translated ENDF files. If 'xdiff' is not available, alter or comment out the line containing the 'xdiff' command.

Note that, while this 'Quick start' will enable translating between formats, more advanced uses of fudge require that you build extensions (written in c or fortran). See below for more detail on making a complete fudge installation.

More detail:

This package is distributed as a zip file. Unzipping produces the directory "fudge-2.0" (note that while GND is currently on version 1.1, the fudge framework has been around longer and is on a later version). Inside the "fudge-2.0" directory are "README.txt", "setup.py", "Makefile", license information, and four sub-directories ("fudge", "bin", "examples" and "doc"). The "Makefile" has several targets, 'clean', 'build' and 'inplace' (the

default target is 'inplace'). These targets facilitate building the full project and cleaning up extra byte-compiled files added when a python source is imported.

After unzipping, change directory into fudge-2.0 and do:

>make (please report any errors or warnings encountered during this step to the developers)

This builds some C extensions that are used to improve performance for some numerically intensive tasks in Fudge.

The package contains a few other files of interest: the new gnd structure is defined inside 'fudge/gnd'. This includes the XML schema, "gnd.xsd", (see below for more about the schema). The "bin" directory contains the main scripts for converting ENDF files to the new format. Users will likely be most interested in the contents of the "bin" and "examples" directories, and in particular the 'rePrint.py' and 'rePrintSample.py' scripts.

The rePrint.py and rePrintSample.py files will now be described.

rePrint.py: This file takes an ENDF file as its first argument and converts it into an internal python representation of the new format. It then writes the new format out in XML to a file named "test.endf6.xml" and rewrites an ENDF version of the new format as the file named "test.endf6". If a second argument is given to rePrint.py, it must be a MT value, and only the reaction channel given by the MT is processed by rePrint.py. Several other files with the prefix "test.endf6." are also outputted by rePrint.py. These files are useful for comparing the original ENDF file to the rewritten ENDF file.

rePrintSample.py: This file randomly picks one of the ENDF files from the fudge2.0/examples directory, calls rePrint.py to process the file and then 'xdiff's the original and rewritten ENDF files (the ones that have been cleansed). Note, it may be necessary to alter or remove the 'xdiff' line if 'xdiff' is missing from one's path, or one would like 'tkdiff', 'kompare' (or whatever) better.

Here are several examples of how to use rePrint.py.

```
python bin/rePrint.py examples/n-026_Fe_056.endf
```

```
python bin/rePrint.py examples/n-026_Fe_056.endf 603 # to process only the MT = 603 data.
```

In case anyone is having problem with the routines in this release, we have included several GND-formatted files in the 'examples' directory. If you are having problems, please let us know.

Nearly all of the neutron, gamma and charged particle sub-libraries from the recent ENDF-VII.1 release can be converted to and from GND using rePrint.py. If you try converting files from other libraries, however, you may encounter errors. Experience has showed us that these errors tend to arise due to problems in the original ENDF file. Typical examples include non-integer multiplicities for neutrons coming from (n,2n), (n,3n) etc, incorrect parent or final levels for gammas, and incorrect Q-values. These are examples of files where the data needs to be fixed by the evaluator. In general, the converter routines will not fix bad data; but instead, will raise an Exception.

If you encounter errors when trying to translate an ENDF file to GND, you may wish to try adding the '--skipBadData' flag to rePrint.py (notice the two leading dashes). If this flag is turned on, when rePrint encounters some common errors it will try simply skipping that MF/MT section from the ENDF file rather than crashing. For example,

```
python bin/rePrint.py --skipBadData ...path_to_ENDF_file.endf
```

Since the errors raised by rePrint.py tend to indicate real format problems with the original ENDF file, we have not made '--skipBadData' the default behavior.

Schema

The XML language has two ways, schema and DTD, to define rules for the structure of an XML file. A validation program uses these rules to verify that an XML file is properly structured (or, if the file can't be validated, to print an informative error message).

A schema is a way to describe the structure of a new format, including the basic hierarchy, required and optional elements and attributes, and the order of the elements. For example, the schema for GND asserts that each file contains one (and only one) 'reactionSuite' element, and that the 'reactionSuite' must contain 'style', 'documentation' and 'particles' elements, and at least one reaction element. We have included a schema file, gnd.xsd (in the 'fudge/gnd' directory), that contains many of the rules for the new format. For more detail on the structure of the new format, please see the example files (in the fudge/examples directory).

There are several benefits to creating an XML schema. First, the schema serves to define the structure of the file, and may be used as the official style guide for GND. Second, we can use the schema to verify whether new evaluations follow the defined format (not just that the XML is well-formed, but also that all required elements are present, and that no unexpected elements appear in the file). Several tools for verifying an xml file against a schema are available. These include the 'xmllint' command-line tool, and also online validation tools. Here are examples of how to use these validation tools:

1) xmllint is a command-line tool built in on Mac OS X and on many Linux distributions (and also available as a download for Windows). To use the 'xmllint' tool to validate a file, do:

```
>xmllint --noout --schema gnd.xsd newEvaluation.gnd
```

```
# the output will be either:
```

```
"newEvaluation.gnd validates"
```

```
# or
```

```
"newEvaluation.gnd fails to validate"
```

2) Validation may also be done using online tools. One useful site is <http://www.xmlvalidation.com/>. To use this online validation, first upload the GND file, and check the box to "Validate against external XML schema " Second, upload the schema. The xml can then be validated against the schema.

Whichever tool is used, xml validation is an easy way to check that the structure of the new evaluation is correct and complete, although more checking will still be required to verify the actual physics content. We also note that, since XML validators can use a schema hosted on the internet, evaluators all over the world can use the same schema. This way we can make sure that the most up-to-date schema is always used.

Please note that 'gnd.xsd' does not yet contain a definition for GND-formatted covariances. While basic covariances can now be translated from ENDF into the new format, the new format is likely to change, and no schema has yet been released.

Sample GND-xml files:

Several examples of the new format are included in this release, in the 'examples' directory.

Converting XML to HDF5

The GND structure is designed to be portable. While rePrint.py produces GND files written in XML, the same structure can easily be expressed in other structured languages. For example, this package includes two examples of the GND-structured files converted from XML to HDF5. HDF5 is a language designed by the National Center for Supercomputing Applications (NCSA) for use in scientific computing (see <http://www.hdfgroup.org/HDF5>). HDF5 features optimized access to very large arrays of data. An HDF5 file may also contain a hierarchical structure very similar to XML. Converting between these two formats is therefore fairly simple.

This package includes a script used to convert from XML to HDF5. Before using the script, one must install:

- a recent version of python (2.4 or higher)
- HDF5, which may be installed manually or via a package manager
- h5py, for reading/writing HDF5 using python. Also available for download or via package manager.

Once these tools are installed, the conversion is simple: (at the command line, type)

```
>python toHDF5.py originalFile.xml
```

A new file titled 'originalFile.hdf5' will be created in the HDF5 format.

Viewing the contents of the HDF5 file is possible using the 'HDFView' application, which is available from the same group that maintains the HDF5 format: <http://www.hdfgroup.org/hdf-java-html/hdfview/>. HDFView permits viewing and modifying data, and can also generate simple 2-d plots.

If you are using an old version of h5py (older than version 1.4), you may find that the HDF5 file is much larger than the original XML. This is

caused by the relatively large disk space required for each 'group' in older forms of HDF5 files. The latest release of HDF5 includes options to substantially shrink the group size, which were implemented in the python interface as of version 1.4.

Issues

1. In MF 6 data, when only discrete photons are present the incident energy interpolation will always be rewritten as 2 (linear-linear), independent of the original file. In this case, no interpolation of the outgoing gamma's energy is needed as it is a constant and the new format does not store an interpolation for this case. When continuum photo data are present, the interpolation is needed. Example showing difference with original file interpolation being 22 (unitbase: linear-linear) and rewritten interpolation being 2 when only discrete gammas are present,

-4.134590-3	-1.641240-2	-1.584840-2	-1.151180-3	1.347590-2	1.783440-2	29428	6	51	231
1.771160-2	1.026730-2	5.075360-3	1.917490-3	4.341750-4		9428	6	51	232
0.000000+0	0.000000+0					29428	6	51	233
2	2					09428	6	51	234
4.482070+4	1.000000+0	2.000000+7	1.000000+0			9428	6	51	235
0.000000+0	0.000000+0					29428	6	51	236
2	22					09428	6	51	237
0.000000+0	4.482070+4					19428	6	51	238
4.463000+4	1.000000+0					9428	6	51	239
0.000000+0	2.000000+7					19428	6	51	240
4.463000+4	1.000000+0					9428	6	51	241
0.000000+0	0.000000+0					09428	6		099999

-4.134590-3	-1.641240-2	-1.584840-2	-1.151180-3	1.347590-2	1.783440-2	29428	6	51	231
1.771160-2	1.026730-2	5.075360-3	1.917490-3	4.341750-4		9428	6	51	232
0.000000+0	0.000000+0					29428	6	51	233
2	2					09428	6	51	234
4.482070+4	1.000000+0	2.000000+7	1.000000+0			9428	6	51	235
0.000000+0	0.000000+0					29428	6	51	236
2	2					09428	6	51	237
0.000000+0	4.482070+4					19428	6	51	238
4.463000+4	1.000000+0					9428	6	51	239
0.000000+0	2.000000+7					19428	6	51	240
4.463000+4	1.000000+0					9428	6	51	241
0.000000+0	0.000000+0					09428	6		099999

2. For multiplicities that are clearly integers (for example (n,n') or (n,2n)) the new format only stores the integer value. The multiplicity in the rewritten file will contain only two energy points as given by the minimum and maximum energy values in the cross section data. Example showing multiplicity difference with original file containing 3 energy point and the rewritten containing only 2,

9.123300+4	2.310380+2	0	2	2	09137	6	18	1	
1.000000+0	1.000000+0	0	1	1	39137	6	18	2	
3	2	0	0	0	09137	6	18	3	
1.000000-5	1.000000+0	1.000000+2	1.000000+0	6.000000+7	1.000000+0	9137	6	18	4
0.000000+0	0.000000+0	1	2	1	879137	6	18	5	
87	22	0	0	0	09137	6	18	6	
0.000000+0	1.000000-5	0	1	225	759137	6	18	7	

9.123300+4	2.310380+2	0	2	2	09137	6	18	1
1.000000+0	1.000000+0	0	1	1	29137	6	18	2
2	2	0	0	0	09137	6	18	3
1.000000-5	1.000000+0	6.000000+7	1.000000+0		9137	6	18	4
0.000000+0	0.000000+0	1	2	1	879137	6	18	5
87	22	0	0	0	09137	6	18	6
0.000000+0	1.000000-5	0	1	225	759137	6	18	7

Second example: even when the original file only contains only two energy points for the multiplicity, the energy values may not exactly align with the cross section minimum and maximum energy values used by the new format,

0.000000+0	0.000000+0	0	0	0	02525	6	099999	
2.505500+4	5.446610+1	0	2	2	02525	6	59	1
1.000000+0	1.000000+0	0	2	1	22525	6	59	2
2	2	0	0	0	02525	6	59	3
2.238360+6	1.000000+0	2.000000+7	1.000000+0		2525	6	59	4

0.000000+0	0.000000+0	0	0	0	02525	6	099999	
2.505500+4	5.446610+1	0	2	2	02525	6	59	1
1.000000+0	1.000000+0	0	2	1	22525	6	59	2
2	2	0	0	0	02525	6	59	3
2.238350+6	1.000000+0	2.000000+7	1.000000+0		2525	6	59	4

3. Some ENDF evaluators do not adhere to the format specifications. As example, for some MF = 3 data, the evaluator will set the fourth value of the first list (which is a reserved value and should always be 0) to a non-zero value. The converter routine does not check reserved fields.

0.000000+0	0.000000+0	0	0	0	06337	3	099999
6.315500+4	1.535900+2	0	1	0	06337	3	51 1
0.000000+0	-7.860000+4	0	0	1	516337	3	51 2
51	2	0	0	0	06337	3	51 3

0.000000+0	0.000000+0	0	0	0	06337	3	099999
6.315500+4	1.535900+2	0	0	0	06337	3	51 1
0.000000+0	-7.860000+4	0	0	1	516337	3	51 2
51	2	0	0	0	06337	3	51 3

4. The new format only stores each particles mass once; where as, an ENDF file may store a particle's mass in many places and the values do not always agree. The ENDF to the-new-format translator uses the first mass it encounters; thus, the original and rewritten mass may not always agree. As example, the first diagram below is part of the header MF, MT = 1, 451 data that list the mass of the target as 112.925, while in the MF, MT = 3, 16 section the target's mass is listed as 112.924 in the original file and 112.925 in the rewritten file,

					1	0	0	0
5.011400+4	1.129250+2	1	0	0	15031	1451	1	
0.000000+0	0.000000+0	0	0	0	65031	1451	2	
1.000000+0	2.000000+7	0	0	10	75031	1451	3	

0.000000+0	0.000000+0	0	0	0	05031	3	099999
5.011400+4	1.129240+2	0	0	0	05031	3	16 1
-1.030410+7	-1.030410+7	0	0	1	155031	3	16 2
15	2	0	0	0	05031	3	16 3

0.000000+0	0.000000+0	0	0	0	05031	3	099999
5.011400+4	1.129250+2	0	0	0	05031	3	16 1
-1.030410+7	-1.030410+7	0	0	1	155031	3	16 2
15	2	0	0	0	05031	3	16 3

5. At times, the original ENDF file will contain floats close but slightly less than 1 (e.g., 0.999994) that are written with a different format than other floats. That is, most floats in the original file, and all in the rewritten file, are produced using a format specification equivalent to the FORTRAN format (1pe11.6); but occasionally, floats in the original file are written using a format specification equivalent to the FORTRAN format (e11.6). The routine fixZeroLeaders.py should fix this problem most of the time. Here is an example,

2.389684+7	1.836111-8	9.999710-1	2.487222+7	1.753413-8	9.999830-12025	6	5	891
2.584760+7	1.701671-8	9.999910-1	2.682298+7	1.654032-8	9.999950-12025	6	5	892
2.779836+7	1.584176-8	0.999997+0	2.877374+7	1.467203-8	0.999998+02025	6	5	893
2.974912+7	1.288630-8	0.999999+0	3.048066+7	1.023804-8	0.999999+02025	6	5	894
3.096835+7	7.926636-9	0.999999+0	3.145604+7	5.121250-9	0.999999+02025	6	5	895
3.194373+7	3.044480-9	0.000000+0	3.243142+7	7.718326-9	0.000000+02025	6	5	896
3.291911+7	1.193127-8	0.000000+0	3.340680+7	2.064115-8	0.000000+02025	6	5	897

2.389684+7	1.836111-8	9.999710-1	2.487222+7	1.753413-8	9.999830-12025	6	5	891
2.584760+7	1.701671-8	9.999910-1	2.682298+7	1.654032-8	9.999950-12025	6	5	892
2.779836+7	1.584176-8	9.999970-1	2.877374+7	1.467203-8	9.999980-12025	6	5	893
2.974912+7	1.288630-8	9.999990-1	3.048066+7	1.023804-8	9.999990-12025	6	5	894
3.096835+7	7.926636-9	9.999990-1	3.145604+7	5.121250-9	9.999990-12025	6	5	895
3.194373+7	3.044480-9	0.000000+0	3.243142+7	7.718326-9	0.000000+02025	6	5	896
3.291911+7	1.193127-8	0.000000+0	3.340680+7	2.064115-8	0.000000+02025	6	5	897

6. Occasionally, an ENDF file will contain multiple discrete gammas with the same energy for a given incident neutron energy. While this is most likely a bug in the data, no check is currently done to flag this, thus it shows up in the rewritten and the XML files. However, the order of this data in the rewritten file may not be the same as in the original ENDF file.

2.000000+7	4.823980+0					6149	6102	28
0.000000+0	0.000000+0	1	2	1		736149	6102	29
73	22	0	0	0		06149	6102	30
0.000000+0	1.000000-5	98	0	526		2636149	6102	31
7.000000+5	3.803069-4	4.856000+5	1.520100-4	4.620000+5	1.396140-26149	6102		32
4.352000+5	8.866738-5	4.263000+5	1.093640-3	4.220000+5	9.541828-46149	6102		33
4.124000+5	5.206589-4	4.078000+5	1.602730-4	4.055000+5	8.861338-46149	6102		34
3.959000+5	4.511089-5	3.881000+5	1.959500-3	3.797000+5	6.527879-46149	6102		35
3.763000+5	1.726180-4	3.648000+5	1.234360-4	3.644000+5	5.782749-46149	6102		36
3.606000+5	1.223420-4	3.516000+5	9.438688-4	3.414000+5	1.012860-46149	6102		37
3.378000+5	1.033400-3	3.339000+5	2.668319-3	3.330000+5	1.000820-36149	6102		38
3.281000+5	8.821378-4	3.180000+5	2.474279-4	3.141000+5	1.351740-46149	6102		39
3.124000+5	4.109689-4	3.111000+5	2.378290-3	3.089000+5	2.507509-36149	6102		40
3.047000+5	5.622569-3	3.022000+5	5.489699-4	2.971000+5	9.391768-46149	6102		41
2.920000+5	4.800809-3	2.848000+5	1.279690-4	2.842000+5	1.214560-46149	6102		42
2.811000+5	8.942088-5	2.722000+5	1.106590-3	2.717000+5	3.781759-36149	6102		43
2.704000+5	1.143680-3	2.684000+5	9.744458-5	2.642000+5	8.812138-56149	6102		44
2.633000+5	1.363430-4	2.594000+5	1.251320-4	2.583000+5	4.710759-46149	6102		45
2.537000+5	1.602590-4	2.502000+5	1.916120-3	2.474000+5	4.760399-46149	6102		46
2.474000+5	8.846588-4	2.461000+5	1.474820-3	2.410000+5	1.637590-46149	6102		47

2.000000+7	4.823980+0					6149	6102	28
0.000000+0	0.000000+0	1	2	1		736149	6102	29
73	22	0	0	0		06149	6102	30
0.000000+0	1.000000-5	98	0	526		2636149	6102	31
7.000000+5	3.803069-4	4.856000+5	1.520100-4	4.620000+5	1.396140-26149	6102		32
4.352000+5	8.866738-5	4.263000+5	1.093640-3	4.220000+5	9.541828-46149	6102		33
4.124000+5	5.206589-4	4.078000+5	1.602730-4	4.055000+5	8.861338-46149	6102		34
3.959000+5	4.511089-5	3.881000+5	1.959500-3	3.797000+5	6.527879-46149	6102		35
3.763000+5	1.726180-4	3.648000+5	1.234360-4	3.644000+5	5.782749-46149	6102		36
3.606000+5	1.223420-4	3.516000+5	9.438688-4	3.414000+5	1.012860-46149	6102		37
3.378000+5	1.033400-3	3.339000+5	2.668319-3	3.330000+5	1.000820-36149	6102		38
3.281000+5	8.821378-4	3.180000+5	2.474279-4	3.141000+5	1.351740-46149	6102		39
3.124000+5	4.109689-4	3.111000+5	2.378290-3	3.089000+5	2.507509-36149	6102		40
3.047000+5	5.622569-3	3.022000+5	5.489699-4	2.971000+5	9.391768-46149	6102		41
2.920000+5	4.800809-3	2.848000+5	1.279690-4	2.842000+5	1.214560-46149	6102		42
2.811000+5	8.942088-5	2.722000+5	1.106590-3	2.717000+5	3.781759-36149	6102		43
2.704000+5	1.143680-3	2.684000+5	9.744458-5	2.642000+5	8.812138-56149	6102		44
2.633000+5	1.363430-4	2.594000+5	1.251320-4	2.583000+5	4.710759-46149	6102		45
2.537000+5	1.602590-4	2.502000+5	1.916120-3	2.474000+5	8.846588-46149	6102		46
2.474000+5	4.760399-4	2.461000+5	1.474820-3	2.410000+5	1.637590-46149	6102		47

7. The converting script will reduce the number of interpolation regions if possible (i.e., without lost of interpolation). For example, in the following diagram the original data has two interpolation regions while the rewritten data has one. These data are point-wise cross section data stored as pairs of $[E, \sigma(E)]$ where E is the projectile's energy and $\sigma(E)$ is the cross section at E . Since the first two points have the same cross section, flat (the original interpolation) and linear (the new interpolation) product the same interpolated value. The rest is linearly interpolated in both; hence, the two are equivalent.

0.000000+0	0.000000+0	0	0	0	0	925	3	099999
9.019000+3	1.883500+1	0	0	0	0	925	3	16
-1.043100+7	-1.043100+7	0	0	2	14	925	3	16
2	1	14	2	0	0	925	3	16
1.098700+7	0.000000+0	1.100000+7	0.000000+0	1.150000+7	1.500000-3	925	3	16
1.200000+7	4.545000-3	1.250000+7	1.400000-2	1.300000+7	2.400000-2	925	3	16
								5

0.000000+0	0.000000+0	0	0	0	0	925	3	099999
9.019000+3	1.883500+1	0	0	0	0	925	3	16
-1.043100+7	-1.043100+7	0	0	1	14	925	3	16
14	2	0	0	0	0	925	3	16
1.098700+7	0.000000+0	1.100000+7	0.000000+0	1.150000+7	1.500000-3	925	3	16
1.200000+7	4.545000-3	1.250000+7	1.400000-2	1.300000+7	2.400000-2	925	3	16
								5

8. The new format converts MF = 8, LO = 0 data into MF = 8, LO = 1 data. LO = 0 describes the residual and how it and its daughters may decay. LO = 1 data only stores the residual, and not its subsequent decay daughters.

0.000000+0	0.000000+0	0	0	0	0	925	6	099999
0.000000+0	0.000000+0	0	0	0	0	925	0	0
9.019000+3	1.883500+1	0	0	1	0	925	8	16
9.018000+3	0.000000+0	9	0	6	922	925	8	16
6.586201+3	2.000000+0	8.018000+3	1.000000+0	6.341000+5	1.000000+0	925	8	16
0.000000+0	0.000000+0	0	0	0	0	925	8	099999

0.000000+0	0.000000+0	0	0	0	0	925	6	099999
0.000000+0	0.000000+0	0	0	0	0	925	0	0
9.019000+3	1.883500+1	0	0	1	1	925	8	16
9.018000+3	0.000000+0	9	0	0	0	925	8	16
0.000000+0	0.000000+0	0	0	0	0	925	8	099999

9. In the new format, primary gamma energies (gammas produced in free-to-bound transitions as opposed to bound-to-bound transitions) are always stored as the binding energy only; as like ENDF MF 12 data. However, for MF 6 data, the total primary energy is stored in ENDF and this data is often not consistent versus projectile energy for the precision given in the ENDF file. In the translation, the binding energy is determined from the gamma's total energy at the first projectile's energy via the formula given below. During rewrite, the total energy is calculated from the binding energy and the projectile's energy as $E_g = E_b + E_i * (m_t / (m_t + m_p))$ where E_g is the total gamma energy, E_b is the binding energy, E_i is the projectile's energy, and m_p and m_t are the masses of the projectile and target, respectively. Below is an example showing the differences in the original and rewritten total primary gamma energies.

0.000000+0	5.000000+0	42	0	198	999849	6102	175
5.007000+5	1.463850-3	4.600000+5	2.172490-2	4.429800+5	6.499520-59849	6102	176
4.400000+5	3.757130-2	4.375600+5	1.736910-4	4.250000+5	2.349500-89849	6102	177
3.840000+5	5.909440-5	3.795200+5	1.498910-2	3.750700+5	7.652190-49849	6102	178
3.150000+5	6.593500-4	2.980200+5	1.181960-3	2.718400+5	2.452190-29849	6102	179
2.550400+5	2.363930-4	2.345600+5	9.592640-5	2.200000+5	1.389550-59849	6102	180
1.915800+5	1.483820-4	1.449600+5	2.167570-3	1.364500+5	2.397310-59849	6102	181
1.362000+5	0.000000+0	7.371000+4	0.000000+0	6.346000+4	1.229040-49849	6102	182
6.249000+4	5.368200-5	5.804000+4	1.017550-5	5.513000+4	1.105140-49849	6102	183
4.298000+4	3.219250-4	-5.084770+6	0.000000+0	-5.125470+6	5.459980-59849	6102	184
-5.142490+6	0.000000+0	-5.145470+6	3.710150-4	-5.147910+6	0.000000+09849	6102	185
-5.160470+6	0.000000+0	-5.168670+6	3.228260-4	-5.201470+6	0.000000+09849	6102	186
-5.205950+6	0.000000+0	-5.270470+6	0.000000+0	-5.342400+6	0.000000+09849	6102	187
-5.365470+6	0.000000+0	-5.397530+6	0.000000+0	-5.440510+6	7.520630-59849	6102	188
-5.449270+6	0.000000+0	-5.522980+6	0.000000+0	-5.585470+6	0.000000+09849	6102	189

0.000000+0	5.000000+0	42	0	198	999849	6102	175
5.007000+5	1.463850-3	4.600000+5	2.172490-2	4.429800+5	6.499520-59849	6102	176
4.400000+5	3.757130-2	4.375600+5	1.736910-4	4.250000+5	2.349500-89849	6102	177
3.840000+5	5.909440-5	3.795200+5	1.498910-2	3.750700+5	7.652190-49849	6102	178
3.150000+5	6.593500-4	2.980200+5	1.181960-3	2.718400+5	2.452190-29849	6102	179
2.550400+5	2.363930-4	2.345600+5	9.592640-5	2.200000+5	1.389550-59849	6102	180
1.915800+5	1.483820-4	1.449600+5	2.167570-3	1.364500+5	2.397310-59849	6102	181
1.362000+5	0.000000+0	7.371000+4	0.000000+0	6.346000+4	1.229040-49849	6102	182
6.249000+4	5.368200-5	5.804000+4	1.017550-5	5.513000+4	1.105140-49849	6102	183
4.298000+4	3.219250-4	-5.084765+6	0.000000+0	-5.125465+6	5.459980-59849	6102	184
-5.142485+6	0.000000+0	-5.145465+6	3.710150-4	-5.147905+6	0.000000+09849	6102	185
-5.160465+6	0.000000+0	-5.168665+6	3.228260-4	-5.201465+6	0.000000+09849	6102	186
-5.205945+6	0.000000+0	-5.270465+6	0.000000+0	-5.342395+6	0.000000+09849	6102	187
-5.365465+6	0.000000+0	-5.397525+6	0.000000+0	-5.440505+6	7.520630-59849	6102	188
-5.449265+6	0.000000+0	-5.522975+6	0.000000+0	-5.585465+6	0.000000+09849	6102	189

10. Some files contain issues that the converting routines cannot fix. In this case, Python raises an Exception (i.e., the code aborts with a traceback and a the reason for the abort). In the following example, endfFileToGND.py aborts because an invalid interpolation is listed for outgoing gammas.

```
>python ~/apps/fudge2/bin/rePrint.py ../neutrons/n-028_Ni_059.endf
 2 [3, 4] : MF=4, LTT = 1
16 [3, 4, 5] : MF=4, LTT = 0 : MF=5, LF=9
28 [3, 4, 5] : MF=4, LTT = 0 : MF=5, LF=9
32 [3, 4, 5] : MF=4, LTT = 0 : MF=5, LF=9
103 [3]
104 [3]
105 [3]
106 [3]
107 [3]
102 [3, 12, 14, 15] : MF=12 LO=1 : ZAP=0 Traceback (most recent call last):
File "/Users/mattoon1/apps/fudge2/bin/rePrint.py", line 72, in <module>
  x,c = endfFileToGND.endfFileToGND( argv[0], singleMTOnly = MT, toStdOut = Options.verbose )
File "/Users/mattoon1/apps/fudge2/fudge/legacy/converting/endfFileToGND.py", line 2647, in endfFileToGND
  radioactiveData, parseCrossSectionOnly = parseCrossSectionOnly )
File "/Users/mattoon1/apps/fudge2/fudge/legacy/converting/endfFileToGND.py", line 2272, in parseReaction
  readMF12_13( info, MT, MTData, productList, warningList )
File "/Users/mattoon1/apps/fudge2/fudge/legacy/converting/endfFileToGND.py", line 1636, in readMF12_13
  readMF15( info, MT, MTData, continuousGamma )
File "/Users/mattoon1/apps/fudge2/fudge/legacy/converting/endfFileToGND.py", line 1735, in readMF15
  ( interpolation, tab1['interpolationInfo'], MT ) )
Exception: Need to implement piecewise energy here. Interpolations differ, 1 vs [[3, 2]], MF=15, MT=102
```