Comments on an API for GND

Bret Beck

Presented at the OECD/NEA/WPEC SG38

Tokai, Japan

10-Dec-2013

Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94551

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and partly funded by the Nuclear Data Program Initiative of the American Recovery and Reinvestment Act (ARRA).

LLNL-PRES-

Statement of work

Document APIs to read and write the new structure in C, C++, Fortran, and Java. Implement a substantiation of an API that is tested by relevant stakeholders of Product Teams 1&2 above and the developers of processing and testing codes.

API outline

- Initial comments
  - Language comment
  - Writing/reading
  - Unit conversion
- Useful methods (functions)
- Belong the reactionSuite
- A comment on reaction matching/searching
- Issues from Fudge work
  - Floating point
  - Zipped files

Initial comments

- Structures and infrastructure/API are entwined
  - Care must be taken when designing the structure so as not to
    - Make the infrastructure/API too complex
    - Make the infrastructure/API consume too much computational
      - memory
      - time

- Definition of a "code group":
  - A class or classes in Object Orientated Programming (OOP) or set of functions (procedural) for handling a task
  - Examples:
    - Reading/writing
    - Unit conversion Discussed during infrastructure session.
    - Reaction matching. Discussed during infrastructure session.

- Indexing should be 0 based
  - Note, in Fortran can define arrays as "(0:n-1)"
    - Example: real x(0:n-1)

Language comments

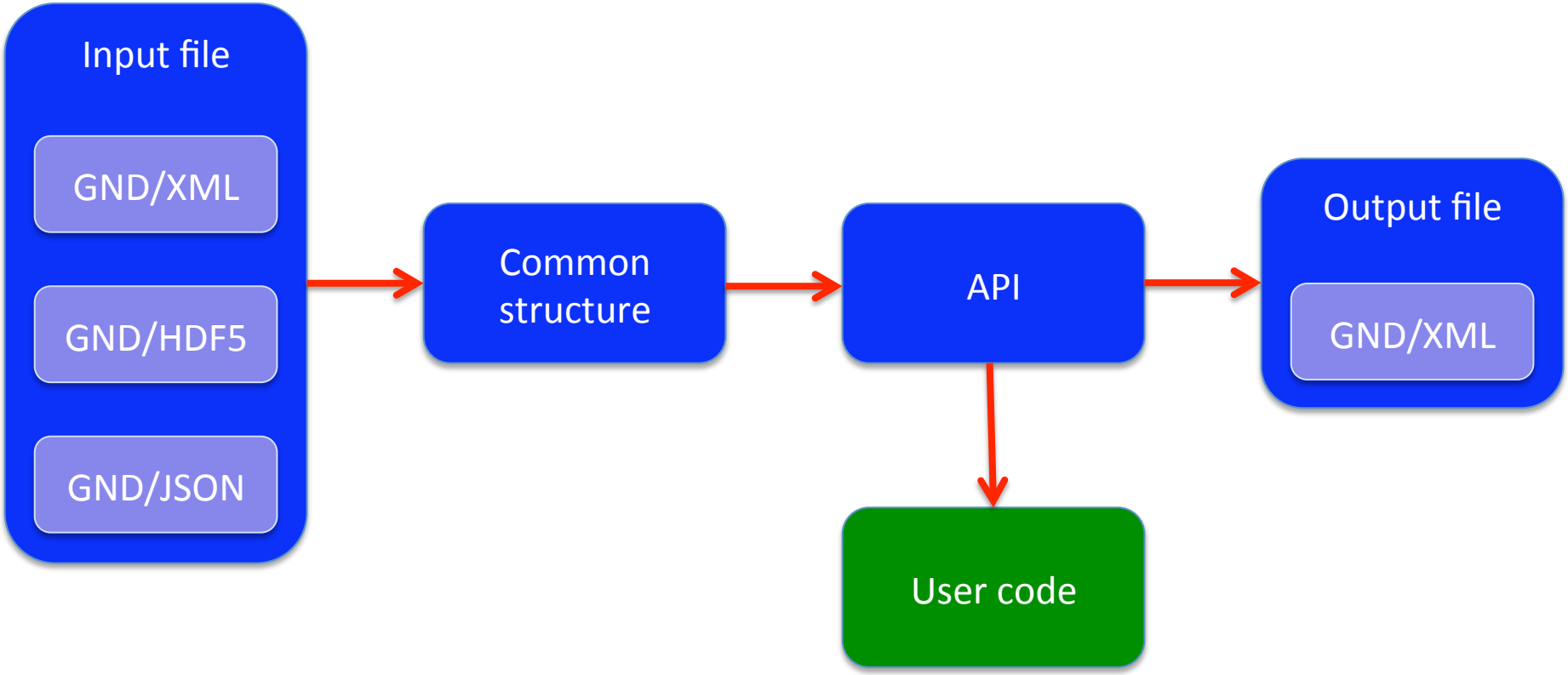
- Procedural languages (e.g., C, FORTRAN-90)
  - Need to define prefixes for all “code groups”
    - That is, all functions (and probably data types) for a code group should have a common prefix
    - Examples: prefixes like gapi_r_, gapi_cs_ and gapi_mul_ for reaction, cross section and multiplicity types and functions.
- Object oriented languages (e.g., C++, FORTRAN-2003)
  - Only need to to define prefixes for classes as each class name prefix its members and methods.
    - C++ example: GOOP_XYs::plot( )
- I think we should write some core groups in C and write wrappers for the other languages.
  - Possible examples: unit conversion and reaction matching.

Only need to support writing to GND/XML format?

- API only needs to support writing to XML
  - Format = Structure + meta-language
    - Written as “Structure/meta-language”
    - Examples: “GND/XML”, “GND/HDF5”
  - Conversion to other formats can be done with codes that only need to know a little of the GND structure, as long as the structure is well defined.
    - Example: In fudge, toHDF5.py is a stand alone python script that converts a “GND/XML” file into “GND/HDF5”. toHDF5.py is roughly 110 lines of python and does not import anything from Fudge.

Need to support reading all formats: GND/XML, GND/HDF5, ...

- Best to create common structure that all formats are converted into



"Common structure" is a standard coding practice – see V. Sinitsa's talk on GRUCON during the infrastructure session

Useful methods (functions)

- Read all/write XML
- Iterate over all items in an object or get an item from an object that are list-like or dictionary like
  - This will be different for OOP and procedural APIs
- convertUnitTo
- Methods to add/delete elements to/from structure
- Get data domain
  - Possible arguments:
    - inUnitOf
- Search/Find
  - Reaction, data type (e.g., cross section), product
- Collapsing of deterministic data?

Useful methods (functions) we may not want in API

- Some things may be handled better by the infrastructure
  - Resonance reconstruction
  - Heating cross sections
  - Linearizing data?
    - toLinear – at least for the last two columns for each data type
      - $\sigma(E)$, for both E and σ
      - $P(\mu|E)$, only for μ and P
      - $P(\mu,E'|E)$, only for μ and P
  - Deterministic grouping of the data

Beyong the reactionSuite

- In GND all reactions for a projectile hitting a target is called a “reactionSuite”.

- Users will want to access various reactionSuite’s.
  - Need an MCNP/ACE/xsdir like structure that contains a list of available reactionSuite’s
  - Need to have a unix ‘ls’ like function to allow users to obtain a list of all reactionSuite’s they request
    - Example
      - findReactionSuites(projectile = ‘n’, target = ‘U*’)
    - Should also define a search/matching expression for this

Another comment on reaction searching/matching

- Should define a set of standard types of search patterns
  - Example
    - rsre_two_n
    - rsre_fission
    - rsre_capture
  - I think these would be better than strings like “capture”, for example
    - Word of caution: In LLNL’s ENDL rsre_capture (or “capture”) would often return two reactions
- Similarly for projectile/target matching

Some issues

Floating point precision

- Is ‘a – a’ zero?
- Not always with finite floating-point precision

```
>>> 1e-5 * 1e-6 / 1e-11 - 1
2.220446e-16
```

- We probably need to define fuzzy comparisons
  - For example, a and b are equal if

$$abs(a - b) <= epsilon * max(abs(a), abs(b))$$

  - Example python ‘compare’ method

```
def compare( value1, value2, epsilon = 0 ) :

    epsilon *= sys.float_info.epsilon
    delta = value1 – value2
    if( abs( delta ) <= epsilon * max( abs( value1 ), abs( value2 ) ) ) : return( 0 )
    if( delta < 0. ) : return( -1 )
    return( 1 )
```

zip-a-dee-doo-dah

- Should the API and infrastructure need to read zipped GND files?
- How about reading only sections of a GND file?
- What should be done in the user code and what can also be done in the API?
  - Should deterministic collapsing be supported?

Comment on pointwise/piecewise data structure

- How could we store (x_i,y_i) pairs of data (e.g., cross section or multiplicity data)?
  - How do we define iterating over the data
  - How do we define indexing the data
- For example, the points (1, 1), (2, 2), (3,1) and (4,0).
  - In python we could define the list xys = [(1, 1), (2, 2), (3,1) and (4,0)]
  - For xys Iteration is

```
for x, y in xys :
```

  - For xys indexing is

```
x, y = xys[index]
```

  - This does not tell us how to interpolate between pairs.
  - Take with same points with ENDF interpolation “2 5 4 2”.
    - Between first two points is log-log while the rest are lin-lin

Comment on pointwise/piecewise data structure - continue

- How we store xys = [(1, 1), (2, 2), (3,1) and (4,0)] in GND
  - With ENDF interpolation '4 2'
    - Stored as an instance of XYs class
    - Allows iteration and indexing in a 'natural' way (e.g., x, y = xys[index])
  - With ENDF interpolation '2 5 4 2'
    - Stored as an instance of regionsXYs class.
    - For this example, the regionXYs instance contains two instances of the XYs class
      - First region: xys1 = [(1, 1), (2, 2)] with 'log-log' interpolation
      - Second region: xys2 = [(2, 2) , (3,1) and (4,0)] with 'lin-lin' interpolation
    - Indexing a regionsXYs instance with index = 0 gives the first region, not the first point.
    - regionXYs class has a 'toPointwise_withLinearXYs' method that converts the points into one XYs instance with 'lin-lin' interpolation.

LLNL's current effort

- GIDI
  - General Interaction Data Interface
  - Written in C
  - C++ wrapper for GEANT
  - Status
    - Only read: no writing
    - Also samples from data

The End