OAK RIDGE National Laboratory

Status of GNDS support in AMPX

D. Wiarda

J. Brown

J. McDonnell

C. Chapman

Oak Ridge National Laboratory

WPEC, May 2022

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

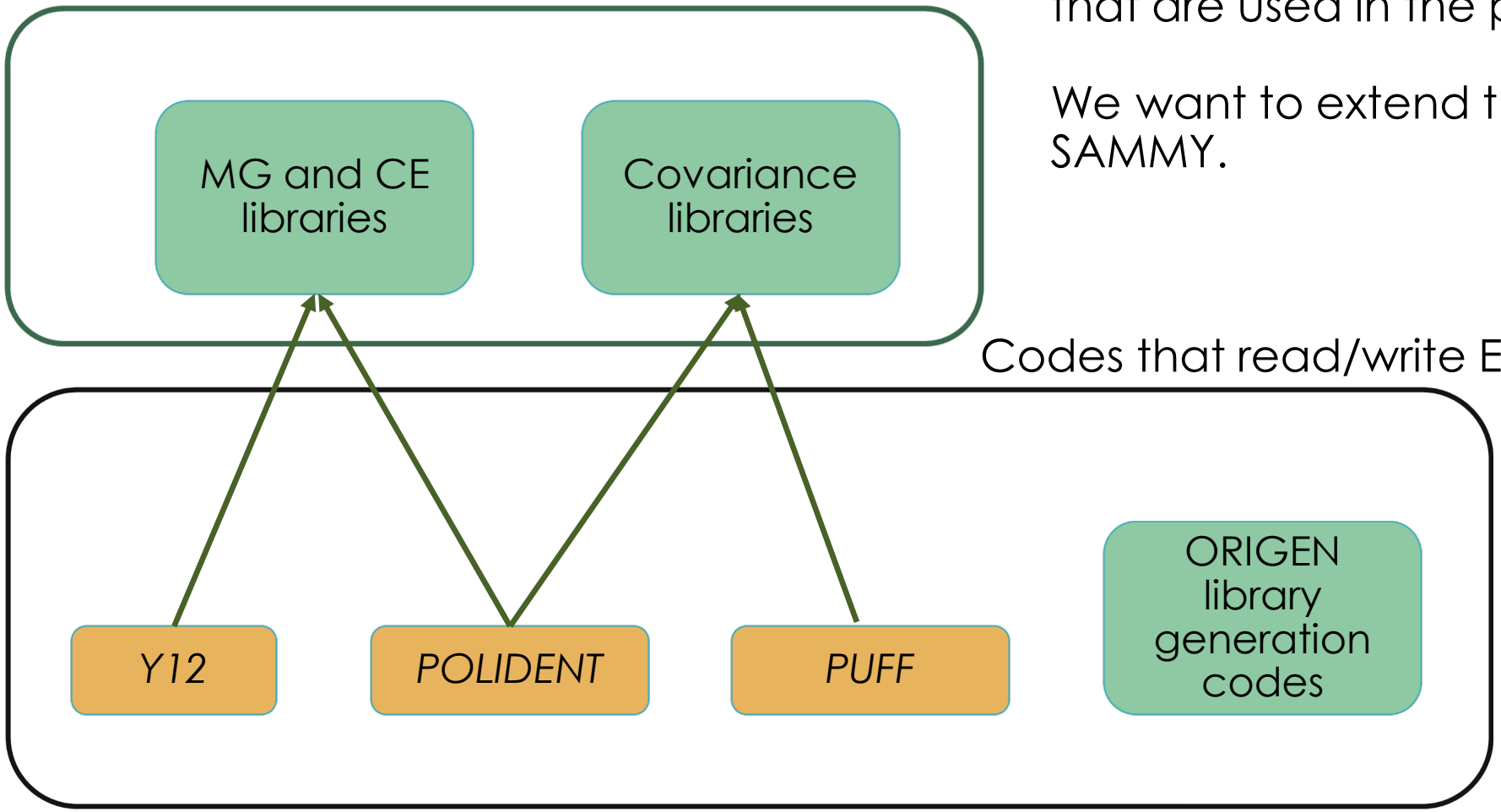
U.S. DEPARTMENT OF ENERGY

Summary

- Overview about ENDF reading in AMPX
- Low level access classes for GNDS used in AMPX
- Use of these classes in AMPX and SAMMY

ENDF data in AMPX

Codes producing final libraries
using SCALE and AMPX in-memory formats



No processing code in AMPX directly accesses the ENDF files. An ENDF reading routine fills in-memory structures that are used in the processing codes.

We want to extend the same process to SAMMY.

Other codes use AMPX/SCALE internal data formats

How to support GNDS

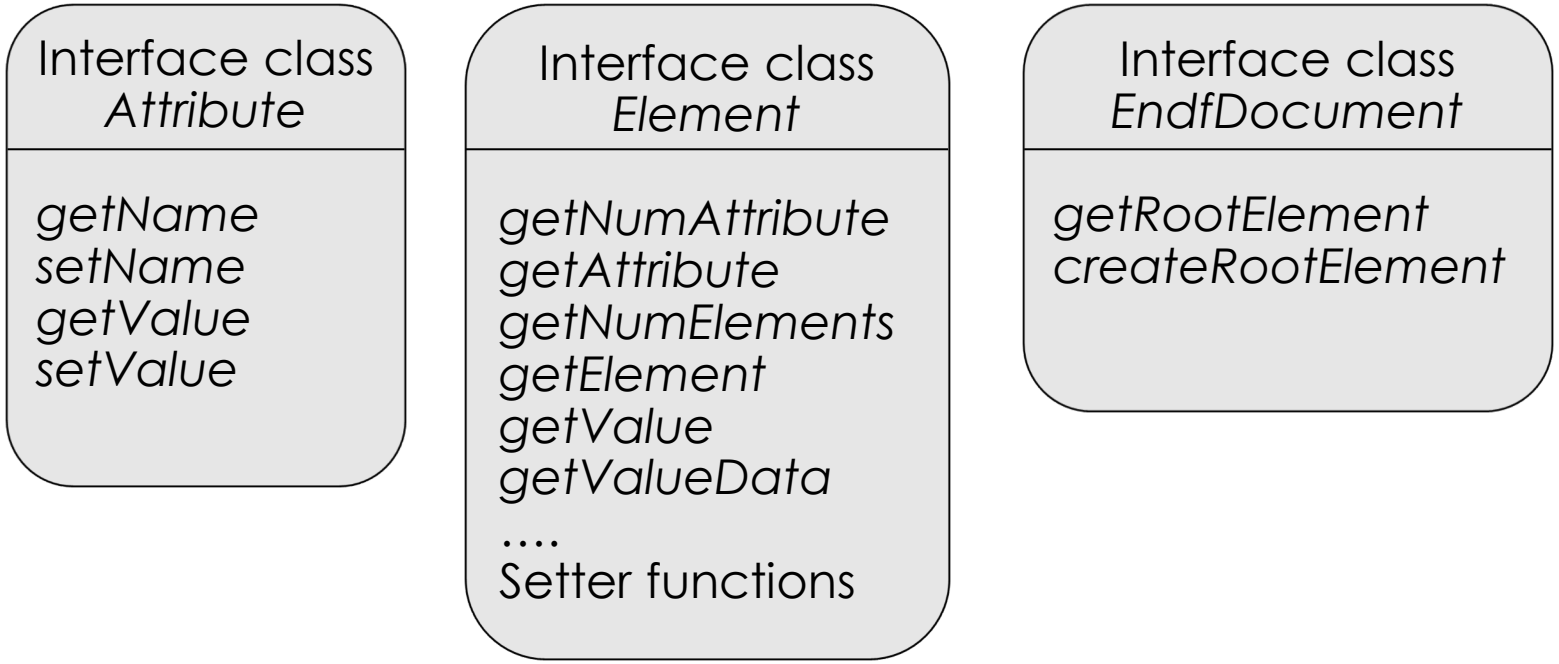
- Internal AMPX C++ structures are the "API" that AMPX and SAMMY will access.
- Add a Reader/Writer that supports GNDS and fills AMPX internal C++ structures.
- Test by processing ENDF formatted and GNDS formatted files and compare results.
- Find an efficient way to support GNDS which also allows us to easily apply updates.
- Updates might not immediately be propagated to the internal AMPX C++ classes if not needed in AMPX.

GNDS low access classes

- Several low-level access classes are used to access the GND files.
- Code has been updated to work with the GNDS-2.0 branch in the NEA GNDS gitlab
- Code is available from https://code.ornl.gov/RNSD/gnds/
- Branch for GNDS-2.0 is at: https://code.ornl.gov/RNSD/gnds/-/tree/GNDS-2.0

On-Disk file access

The direct access to the XML (or HDF5 or JSON) GNDS file is abstracted into access to elements and attributes:



Based on pugixml. This is a layer on top of the actual XML DOM reader to allow easy switch to different reader if needed. Classes are in *src/xml*

Parent classes for all GND classes

- ***EndfDocument***
  - The abstract class to allow access to the elements and attributes of the GNDS file to shield the downstream classes from on-disk format
- ***Defintions***
  - Basic conversion roles from text to numbers
  - Enumerations for Interpolation rules
- ***Container***
  - Establishes the tag name and keeps track of the *label* and *value* attributes
  - Has the base implementation to retrieve/cache element information from disk

Classes are in *src* directory

Classes for selected GPDC objects

In order to make access more fine-grained a couple of low-level GNDS data container have hand-edited C++ source code:

- ***ValuesContainer*** (for "gpdc::values" child)
- ***TableContainer*** (for "gpdc::table" child)
- ***ArrayContainer*** (for "gpdc::array" chid)
- ***ExternalFiles*** (for "gpdc::ExtrernalFiles" and "gpdc::ExtrernalFile" children)

Most GNDS classes inherit from ***GNDElement*** to manage references and external files.

Classes are in *src* directory

GNDS specification classes

The remaining classes are generated from the JSON definition files, using python classes:

- ***GenerateFromJson*** and helper files

Input is:

- File containing the JSON files to parse (relevant file for GNDS-2.0 is given as ***FormatJsonFile*** in ***FromJson*** directory)
- The directory containing the GNDS-2.0 gitlab data
- Final directory for the generated classes

Generated classes are available in ***src/gnd*** directory.

Note on python generator classes

- C++ classes will automatically be generated for all tags listed in the specified JSON definition files.
- Tags like "table" or "array" will immediately reference the hand-coded classes instead of generating new classes.
- Some resolution for ambiguous tags is attempted and reported (see next slide).
- Some tags need to have special C++ names, as the tag name is a reserved name ('double' -> 'gndsDouble')
- JSON namespace become C++ namespaces
- The project has a CMakeLists.txt file that allows to build a library. This library is directly built and used in our SCALE builds.

Example output for ***GenerateFromJson***

```
bash-3.2$ python GenerateFromJson.py  FormatJsonFile ~/fudge/formats/ ../src/gnd/
Reading: /Users/dw8/fudge/formats//Styles/summary_styles.json
…
Reading: /Users/dw8/fudge/formats//Pops/summary_pops.json
mass refered to from tsl was found in common and pops
   Disregarding the pops version and use common
  Failed to  find xs in  context gpdc
     using xs_in_xs_pdf_cdf1d in gpdc
  Failed to  find cdf in  context gpdc
     using cdf_in_xs_pdf_cdf1d in gpdc
  Failed to  find PoPs in  context resonances
     using PoPs_database in pops
Q refered to from resonances was found in common and pops
   Disregarding the pops version and use common
uncertainty refered to from abstract was found in gpdc and pops
   Disregarding the pops version and use gpdc
  Failed to  find functional in  context abstract
     using functionalNode in abstract
product refered to from fissionFragmentData was found in common and pops
   Disregarding the pops version and use common
energy refered to from processed was found in common and pops
   Disregarding the pops version and use common
energy refered to from processed was found in common and fpy
   Use the common version
  Failed to  find angular in  context processed
     using angular_uncorrelated in transport
products refered to from transport was found in common and pops
   Disregarding the pops version and use common
  Failed to  find PoPs in  context transport
     using PoPs_database in pops
```

Note on python generator classes cont.

All data classes like ***XYs1D***, ***regions1D***, etc now contain an "*abstractNode*" listed as "*functional*" or "*functionalNode*" in the GNDS definitions.

- The python code checks that it is in namespace gpdc and that is one of the classes denoted below

- The content of "*function1ds*" is rewritten as having children:
  - '*XYs1d*', '*regions1d*', '*gridded1d*', 'Ys1d', '*Legendre*' , '*polynomial1d*', '*constant1d*', '*xs_pdf_cdf1d*'
- The content of "function2ds" is rewritten as having children:
  - '*XYs2d*', '*regions2d*', '*gridded2d*'
- The content of "function3ds" is rewritten as having children:
  - '*XYs3d*', '*regions3d*', '*gridded3d*'

Generated C++ classes

- The classes, by virtue of being generated directly from the JSON specifications are very close to the GNDS specification
- Children are:
  - One shared pointer to the child class if occurrence is one or optional
  - Vector of shared pointer if occurrence is more than one
- Classes allow to read and write GNDS formatted files.
- ToDo: We need to add more unit tests for all the classes.

GNDS access layers in AMPX (cont.)

Classes that fill the AMPX C++ in-memory structures. These classes are needed as:

- To select the correct "style" of the data the user requested, which includes following the inheritance chain.
- Convert GNDS units to AMPX units
- Convert GNDS constructs into AMPX constructs.
- More user-friendly access methods to Particle data base

This layer is currently only reading data, but writing will be added as needed. The first implementation for reading will be for resonance parameters and corresponding covariance matrices for use in SAMMY.

Classes for GNDS-1.9 are available in SCALE 6.3 release and they are currently updated as needed for GNDS-2.0.

SAMMY/AMPX and GNDS – future plans

- SAMMY has its own ENDF reading and writing routines.
- **We plan on switching to the AMPX reading and writing routines.** This was delayed in favor of using in-memory C++ AMPX classes for resonance parameter and all covariance information.
- Having the relevant information in the in-memory classes will make it much easier to switch out the reading and writing to use AMPX methods.
- AMPX is in the final stages to add support for reading GNDS formatted ENDF files.
- **Switching to these routines will bring GNDS support to SAMMY.** In this context, writing of GNDS files will also be added.

SAMMY AND AMPX Release

- SAMMY source code is available from https://code.ornl.gov/RNSD/SAMMY
- The code currently needs SCALE 6.3 beta 9 and up to compile, instructions are provided.
- AMPX is part of SCALE and available with SCALE 6.2 and up. Can be requested from the NEA Databank and RSICC.
- We are allowed to distribute AMPX as open source but have not yet completed the correct mix of decoupling and sharing with SCALE.

This work was supported by the Nuclear Criticality Safety Program, funded and managed by the National Nuclear Security Administration for the Department of Energy.