NNDC ENDF I/O Library

S. Hoblit
National Nuclear Data Center
Brookhaven National Laboratory

hoblit@bnl.gov

Brookhaven Science Associates

Motivation & History

- NNDC had existing fortran95 routines for reading/writing MF3/33, etc.
  - Individual routines for specific files
  - array size usually hardcoded
- Evolved (as needed) to read other MF sections
  - Add MF2, MF33/34, ..
- Assemble into library of I/O routines
  - Unify interface for various MF sections
- Extend support to entire ENDF format

Why now?

- With the new GND format being developed, why add library for a dying format?
  - Move from a 'file-centric' to a 'API-centric' model. Application code insulated from “crazy aunt in the attic” ENDF format. This API approach provides a level of abstraction that can be applied to other formats. Code written to API interface; let I/O library deal with details of file format.

Device driver analog

User code

User code

Device driver

Abstraction layer

I/O library

Physical layer

ENDF DATA FILE

Organization

- Collect all I/O for ENDF in common library
  - Changes to format can be addressed with modifications to library. Processing code not burdened with details of ENDF file structure.
- Use F95 modules to define data types and supply the routines that operate on them.
  - Dynamically allocated types, memory allocated as needed, no preset limits. ENDF data types hold an entire ENDF file. Simplify processing codes.

Basic ENDF data types

```
type endf_file
    character*80 hdline                    ! “header” line
    type (endf_mat), pointer :: mat        ! materials
end type

type endf_mat
      integer mat
      type (endf_mat), pointer :: next
      type (MF_1),  pointer :: mf1
      type (MF_2),  pointer :: mf2
      type (MF_3),  pointer :: mf3
      type (MF_4),  pointer :: mf4
      type (MF_5),  pointer :: mf5
      type (MF_6),  pointer :: mf6
     ...
      type (MF_32), pointer :: mf32
      type (MF_33), pointer :: mf33
      type (MF_34), pointer :: mf34
      type (MF_35), pointer :: mf35
      type (MF_40), pointer :: mf40
end type
```

Simple MF type - MF3

```
type MF_3
    type (MF_3), pointer :: next
    integer mt                  ! MT reaction
    integer lc                  ! line count
    real za                     ! ZA
    real awr                    ! AWR
    real qm                     ! mass-diff Q-value
    real qi                     ! Q-value for lowest E state in MT = QM-E_excited
    integer lr                  ! "complex" breakup flag
    type (tab1) tb              ! table of values
end type


type tab1
    integer nr                                  ! # NR interpolation ranges
    integer np                                  ! # points
    type (int_pair),  pointer :: itp(:)         ! interpolation tables
    type (real_pair), pointer :: dat(:)         ! data values
end type
```

Covariance example

```
type MF33_sect
    real xmf1                                   ! mf of 2nd E-dep crs
    real xlfs1                                  ! final excited state of 2nd E-dep crs
    integer mat1                                ! MAT for 2nd E-dep crs
    integer mt1                                 ! MT for 2nd E-dep crs
    integer nc                                  ! # of NC-type sub-sections
    integer ni                                  ! # of NI-type sub-sections
    type (nc_cov_sect), pointer :: ncs(:)     ! NC sections
    type (ni_cov_sect), pointer :: nis(:)     ! NI sections
end type
```

```
type MF_33
     type (MF_33), pointer :: next
     integer mt
     integer lc
     real za
     real awr
     integer mtl                                ! MT for lumped covar (851-870)
     integer nl                                 ! number of sections (with same MT)
     type (MF33_sect), pointer :: sct(:)        ! sections (nl)
 end type
```

More complicated example - MF8

```
type mf8_branch
   sequence
   real hl                    ! half-life (sec)
   real rtyp                  ! decay type
   real zan                   ! ZA of next nuclide in chain
   real br                    ! branching ratio for ZAN & level
   real end                   ! endpoint energy of particle produced
   real ct                    ! chain terminator (see ENDF manual)
end type

type mf8_nucprod
   real zap                   ! ZA of nuclide produced
   real elfs                  ! energy of state produced (eV)
   integer lmf                ! MF where multiplicity is found
   integer lfs                ! level number state formed
   integer nd                 ! # branches into which nuclide decays
   type (mf8_branch), pointer :: br(:)    ! branches (nd)
end type

type mf8_nuclide
   integer lis                ! state # of target ZA
   integer liso               ! isomeric state # of target
   integer ns                 ! # states of reaction product where decay data given
   integer no                 ! flag denoting location decay data. 0=here, 1=MT=457.
   type (mf8_nucprod), pointer :: prd(:)
end type

type mf8_fp_yield_data
   sequence
   real zafp                  ! ZA for fission product
   real fps                   ! state # for fission product
   real y                     ! MT454: yi=yield for FP prior to decay; MT459 yc=cumulative yield
   real dy                    ! uncert in yi or yc
end type

type mf8_fp_epoint
   real e                     ! incident energy (eV)
   integer itp                ! interpolation table
   integer nfp                ! # product nuclide states
   type (mf8_fp_yield_data), pointer :: c(:)  ! yield data (nfp)
end type

type mf8_fp_yield
   integer le                 ! # incident energies specified
   type (mf8_fp_epoint), pointer :: ep(:)     ! yields at le energies
end type

type mf8_decay_discrete_spectrum
   sequence
   integer nt                 ! # items specifed (either all 12 or just first 6)
   integer dum                ! dummy for alignment
   real er                    ! ave decay energy of radiation produced
   real der                   ! unc in er
   real rtyp                  ! decay mode
   real type                  ! type of trans for e capture
   real ri                    ! intensity of rad produced
   real dri                   ! unc in ri
   real ris                   ! internal pair form coef
   real dris                  ! unc in ris
   real ricc                  ! total internal conversion coeff
   real dricc                 ! unc in ricc
   real rick                  ! k-shell int conv coef
   real drick                 ! unc in rick
   real ricl                  ! l-shell int conv coef
   real dricl                 ! unc in ricl
end type

type mf8_decay_continuous_spectrum
   real rtyp                  ! decay mode
   integer lcov               ! flag for covariance data(0=no, 1=yes)
   type (tab1) rp             ! spectrum of cont component of radiation (prob/eV)
end type

type mf8_decay_covar_spectrum
   integer lb                 ! flag
   integer npp                ! # pairs
   type (real_pair), pointer :: ef(:)    ! E,F pairs
end type

type mf8_decay_data
   integer lis                ! state of original nuclide
   integer liso               ! isomeric state #
   integer nst                ! stability flag
   integer nsp                ! total # of radiation types with spectral info given
   real t12                   ! half-life
   real dt12                  ! error
   integer nc                 ! # decay energies (3 or 17)
   type (real_pair), pointer :: ex(:)  ! decay energies (ex & dex)
   real spi                   ! spin of state lis
   real par                   ! parity of state lis
   integer ndk                ! # decay modes
   type (mf8_decay_mode), pointer :: dcm(:)     ! decay modes (ndk)
   type (mf8_decay_spectum), pointer :: spt(:)  ! decay specta (nsp)
end type

type mf8_decay_spec_data
   sequence
   real fd                    ! discrete spectrum norm factor
   real dfd                   ! unc in fd
   real erave                 ! ave decay energy of radiation produced
   real derave                ! unc in erave
   real fc                    ! continuum spectrum norm factor
   real dfc                   ! unc in fc
end type

type mf8_decay_spectum
   real styp                  ! decay radiation type
   integer lcon               ! continuum spectrum flag
   integer ner                ! # discrete energies given
   type (mf8_decay_spec_data) dat    ! norm factors & energies
   type (mf8_decay_discrete_spectrum), pointer :: dsc(:)   ! discrete spectra data (ner)
   type (mf8_decay_continuous_spectrum), pointer :: con    ! continuum spectra data
   type (mf8_decay_covar_spectrum), pointer :: cov         ! spectra covariance data
end type

type mf8_decay_mode
   sequence
   real rtyp                  ! decay mode
   real rfs                   ! isomeric state flag for daughter nuclide
   real q                     ! total decay energy available
   real dq                    ! unc in q
   real br                    ! frac of decay of nuclide in state LIS into this decay mode
   real dbr                   ! unc in dbr
end type

type MF_8
   type (mf_8), pointer :: next          ! next section
   integer mt                            ! MT
   integer lc                            ! line count
   real za                               ! ZA for material
   real awr                              ! AWR for material
   type(mf8_nuclide),    pointer :: ncl      ! radioactive nuclide prod
   type(mf8_fp_yield),   pointer :: fpy      ! fission product yeilds
   type(mf8_decay_data), pointer :: rdd      ! radioactive decay data
end type
```

External user interface

read_endf_file(filename, endf [,mat])

- **filename:** (output) character string containing name of ENDF file to read
- **endf:** (output) endf data type. Container for ENDF file data.
- **mat:** (input) integer, optional, MAT to read. if present, only read this materials with this MAT number. If not present read entire ENDF file.
- **returns:** integer status code. (success=0)

External user interface

write_endf_file(filename, endf [,overwrite])

- **filename:** (input) character string containing name of ENDF file to write
- **endf:** (input) endf data type
- **overwrite:** (input) logical, optional. If present and .true., then allow overwriting existing file. If not or false, do not allow overwrite.
- **returns:** integer status code. (success=0)

External user interface

del_endf(endf)

- **endf:** (input/output) endf data type.
- **returns:** integer status code. (success=0)

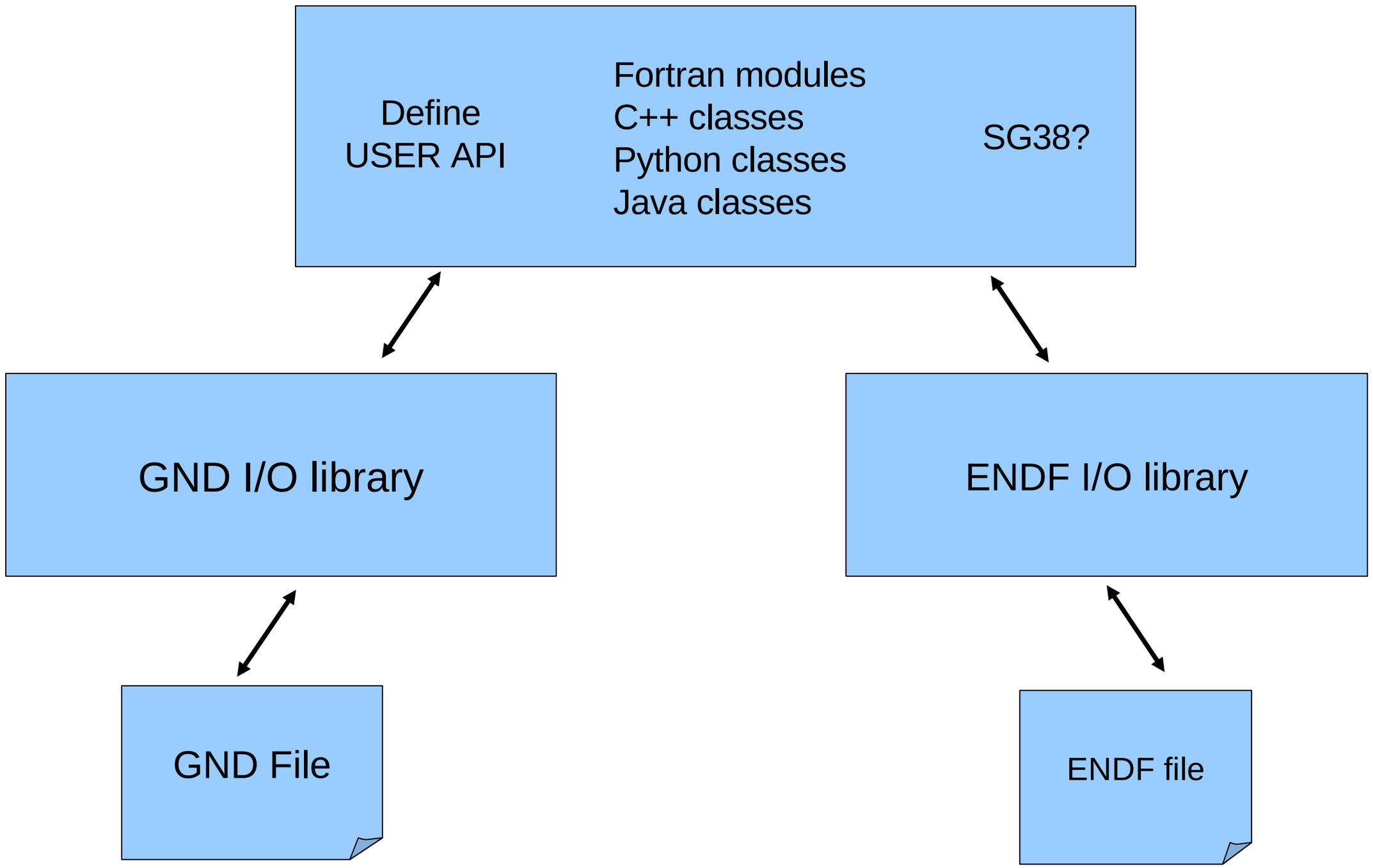
Deallocates an endf data type.

Used when the data held in the endf type are no longer needed. Once deallocated, all memory released and instance available for further reads.

Simple coding example

```
program stanef

use endf_io

implicit none

logical*4 qover
integer*4 nout,nin,status
character*200 outfile, infile

type (endf_file) endf

call parse_cmd_line(outfile,nout,qover,infile,nin)

write(6,*) ' Reading '//infile(1:nin)
status = read_endf_file(infile(1:nin),endf)
if(status /= 0) then
    write(6,*) ' Error reading '//infile(1:nin)
    stop ' No output file written'
endif

call reset_mf1(endf)

write(6,*) ' Writing '//outfile(1:nout)
status = write_endf_file(outfile(1:nout),endf,qover)
if(status /= 0) then
    write(6,*) ' Error writing '//outfile(1:nout)
    stop ' Output file may be incomplete'
endif

end
```

Higher level API?



Summary

- ENDF I/O fortran-95 library – 'gentle' migration of existing fortran-77 code with compatible fortran-95 modules.
- Performs format checking automatically.
- No preset limits – arrays dynamically allocated.
- Provides **API interface**. User code can focus on the ***physics***, not the ***formats***.
- Modular code. Currently supports unix/mac, but all OS dependencies contained in small jacket routine that is easily extendable to other O/S environments (windows).
- Developed by NNDC for NNDC, but source code is available. Any comments/suggestions welcome.