A C++ interface to nuclear structure (ENSDF) data

P. Urkedal
Division of Mathematical Physics
Lund Institute of Technology

2002-03-23, revision 1.2

Abstract

The `libmore` C++ library provides access to experimental nuclear structure data. It is intended for computer algorithms which process the data numerically, rather than for the purpose of presentation. A highly parsed representation of the datasets gives easy and fast access to the data, and a cache mechanism reduces the number of times the data is re-parsed.

1 Introduction

When doing systematic studies of nuclear structure, it is feasible to have automated access to recent experimental data. The Evaluated Structure Data Files (ENSDF) is the source of the well-know Nuclear Data Sheets, and provides detailed and recent data in electronic format. C++ is a widespread language which is suitable for scientific computation. To my knowledge, no library exist today which provides ENSDF data to C++ (or C) programs. The present library was used in [5].

This ENSDF parser is module of the larger `libmore` [1] library, which requires a POSIX compliant operating system in addition to a modern C++ compiler. Some minor debugging may be needed for particular platform and compiler combinations. The author will provide help and coordination of such effort. As follows is a summary of the design of the ENSDF module, and supplements the reference manual that comes with the library. Also, the ENSDF preparation manual [4] may be good to at hand.

2 The design

When referring to identifiers, only the last namespace component is included. (See the top of the sample program in the appendix or the full reference manual for the proper **using** directives.)

The entry point to the datasets is the `ens::nucleus` type. By specifying a nucleus upon construction, it provides an STL [2] iterator range over datasets. Each dataset is a collection of different record types organized as a tree. The records are organized similarly to the structure described in [4]. A dataset (`ens::dataset`) consists of

- Qualitative information like identification of the dataset, revision history, definitions of cross-references used in various parts of the dataset, and comments.
- Possibly a Q-Value record (`ens::rec_qvalue`) which may contain neutron and proton separation energies and Q-values for β^- and α decay.
- Decay information (`ens::decay_info`) contains some records with information about the normalization of decay strengths.
- Levels (`ens::rec_level`). Information for each level, with a sequence of radiations (`ens::rec_radiation`).

- Unplaced radiations (`ens::rec_radiation`).

Comments are attached as sub-nodes of the records they comment upon. There are four kinds of radiation records (inherited from `ens::rec_radiation`),

- The `ens::rec_gamma` record contains information on a γ decay.
- In a reaction dataset of the daughter nucleus, `ens::rec_beta_mi` contains information on a β^- decay and `ens::rec_beta_pl` contains information on an electron capture or β^+ decay.
- Also in the datasets of the daughter nucleus, `ens::rec_particle` contains information on various of particle or delayed particle decays.

The precise details of each data structure is described in the `libmore` reference manual.

The appendix shows a complete sample program. In line 19 the the entry point `ens_nucl` to the datasets is defined. In lines 22 to 33, a dataset with identification string containing the substring "EC DECAY" (electron capture) is searched for. Line 36 iterates over the levels of the dataset, and line 41 and 47 iterates over radiations and selects the record type which contains electron capture and β^+ decay data.

Values with various kinds of uncertainties are very common in the ENSDFs. There are symmetric and asymmetric confidence intervals. There are numbers which are upper or lower bounds, and numbers with unspecified standard deviations. Values may be tagged as deriving from systematics or model-dependent calculations. We have encapsulated this information as a type `ens::confiv_t` (which is a typedef of a template type). The type specifies a "central" value and, relative to this, the upper and lower bounds of a confidence interval. Each of these numeric parts can be tagged as unknown or infinite. A known central value with an infinite upper (lower) deviation specifies the lower (upper) bound on the physical quantity. The rest of the `ens::confiv_t` are small bitfields representing the origin of the quantity and some additional flags. A configurable printing subroutine is provided. For example, the program from the appendix with argument `Ca44` gives

```
1.157039(15)(ex)      log10(f T[1/2]) = 5.300000(ex)
2.656530(24)(ex)      log10(f T[1/2]) = 5.160(10)(ex)
  3.30146(6)(ex)      log10(f T[1/2]) = 6.60(10)(ex)
    3.307900(ex)      log10(f T[1/2]) = 7.20(10)(ex)
```

All physical quantities are provided by `libmore` in SI units, independent of the unit used in the ENSDF. For instance, the SI unit J is used for energy rather than keV, which used in the ENSDFs, or MeV. The idea is that it is easier to detect conversion errors between J and MeV than between keV and MeV. Besides, consistent use of SI units means that the same definitions of physical constants can be used for both sub-atomic and macroscopic applications. Due to the wide range of the exponent of modern double precision datatypes, the smallness of the energies should not cause any numerical problems, even when J is used consistently in the application. Values given in percent in the ENSDF is divided by 100 to give a share of the total.

When a dataset is requested, `libmore` will look for the datasets for that nucleus on the local machine. Else, if there was no prior attempt to fetch the data, it will be fetched from a remote site, compressed and stored on the local disc indefinitely (or until the user removes the entry). The dataset is then parsed and stored in a memory cache. Unused entries are removed from the memory cache when there are sufficiently many of them; the exact mechanism is subject to changes.

3 Outstanding issues

Continuation records are ignored at the moment. There are also parse errors on some of the ENSDF files. The deep parsing of numeric quantities and the expressive syntax of many of the ENSDF fields make it difficult to create a perfect parser. Finally, there should be a way to query the ENSDF server of for new entries without actually fetching the files. Also, a major part of the network traffic can also be saved by compressing the data before transport. The `bzip2` [3] utility gives a compression factor better than 6 for the average ENSDF file.

Appendix. A sample program

The following program takes a nucleus as its argument, and prints out the energy levels and the $\log_{10}(f\tau_{1/2})$ for electron capture and β^+ decays to this nucleus.

```
#include <more/io/cmdline.h>
#include <more/phys/ens.h>

using more::io;    // IO namespace including the command-line parser.
using more::phys; // Physics namespace
using phys::ens;  // The ENSDF data structures
using phys::si;   // SI units

int
main(int argc, char** argv)
try {
    // Parse command-line arguments.
    io::cmdline cl;
    phys::nucleus nucl;
    cl.insert_reference("", "X#A␣nucleus.", nucl);
    cl.parse(argc, argv);

    // Obtain the datasets.
    ens::nucleus ens_nucl(nucl);

    // Find the dataset for electron capture and beta+
    ens::nucleus::dataset_iterator it_ds
        = ens_nucl.dataset_begin();
    while (it_ds != ens_nucl.dataset_end()) {
        std::string idn = it_ds->ident()->dataset_id_string();
        if (idn.find("EC␣DECAY") != std::string::npos)
            break;
        ++it_ds;
    }
    if (it_ds == ens_nucl.dataset_end()) {
        std::cout << "There␣is␣no␣EC␣dataset␣for␣this␣nucleus.\n";
        return 0;
    }

    // Each dataset may contain a range of levels for the nucleus
    for (ens::dataset::level_iterator it_lev = it_ds->level_begin();
         it_lev != it_ds->level_end(); ++it_lev) {

        // The level may contain a range of radiation records for
        // decays with this level as the _final_ state.
        for (ens::rec_level::radiation_iterator
                 it_rad = it_lev->radiation_begin();
             it_rad != it_lev->radiation_end(); ++it_rad) {

            // We are only interested in the rediation if it comes
            // form beta decay.
            if (ens::rec_beta_pl const* beta_pl = it_rad->to_beta_pl()) {

                // Print the energy of the level.  This energy may be
                // specified relative to some reference point.
                std::cout << std::setw(24)
```

```
                          << it_lev->E_minus_E_ref()/si::MeV;
                ens::print_energy_ref(std::cout, it_lev->i_E_ref());

                // Print the log(f T[1/2]) value.
                std::cout << "log10(f␣T[1/2])␣=␣"
                          << beta_pl->log_ft() << '\n';
            }
        }
    }
    return 0;
}
catch (io::cmdline::relax) { // Cf. more::phys::cmdline (option --help)
    return 0;
}
catch (std::exception const& xc) {
    std::cerr << "**␣exception:␣" << xc.what() << '\n';
    return 1;
}
```

References

[1] Hosted at `http://more.sourceforge.net`.

[2] *Programming languages — C++*, 1998. ISO/IEC-14882.

[3] Julian Seward. `http://sourceware.cygnus.com/bzip2`, `http://www.muraroa.demon.co.uk`.

[4] Jagdish K. Tuli. *Evaluated Nuclear Structure Data File: A Manual for Preparation of Datasets*. National Nuclear Data Center, Brookhaven National Laboratory, P.O. Box 5000, Upton, New York 11973-5000, 2001.

[5] P. Urkedal. Systematic study of Gamow-Teller transitions. To be submitted to Phys. Rev. C.