Webnucleo Technical Report: libnucnet Iterators

Bradley S. Meyer

July 21, 2008

This technical report describes how to iterate over species, reactions, or zones with libnucnet and how to apply functions during the iterations.

1 Iterations

Species, reactions, and zones are stored in hashes and lists in libnucnet. As of version 0.2 of libnucnet, a user loops over them by calling an iterator and supplying an iterate function to apply during the iteration.

2 Iterating over Nuclear Species

To iterate over the species in a collection of species stored in a Libnucnet__Nuc structure, the user first writes a routine to apply to each species during the iteration. The prototype is

```
int
my_iterate_function(
  Libnucnet__Species *p_species,
  void *p_my_data
);
```

The routine may have any appropriate name. The necessary inputs are:

- **p_species:** A pointer to a species in the collection.
- **p_my_data:** A pointer to a user-defined data structure containing extra data to be passed to the user's iterate function.

The routine must return 1 to continue iteration or 0 (zero) to stop.

The user then calls the function to apply with the Libnucnet__Nuc API routine Libnucnet__Nuc__iterateSpecies() routine, which has the prototype

```
void
Libnucnet__Nuc__iterateSpecies(
  Libnucnet__Nuc *self,
```

```
  Libnucnet__Species__iterateFunction pf_function,
  void *p_user_data
);
```

The necessary inputs are:

- **self:** A pointer to a Libnucnet__Nuc structure, which contains the collection of nuclear species.
- **pf_function:** The name of the user's function to be applied during the iteration. Typically, this needs to be cast as a Libnucnet__Species__iterateFunction.
- **p_my_data:** A pointer to a user-define data structure containing extra data to be passed to the user's iterate function. If there are no extra data, this should be NULL.

For example, suppose we want to count the number of species with $Z \geq 10$ in the Libnucnet__Nuc structure pointed to by p_my_nuclei and print out their name. We first write an iterate function, which we will call my_counter_and_printer:

```
int
my_counter_and_printer(
  Libnucnet__Species *p_species,
  int *p_count
)
{

  if( !p_species || !p_count )
  {
    fprintf( stderr, "Problem with species or user data.\n" );
    return 0;
  }

  if( Libnucnet__Species__getZ( p_species ) >= 10 )
  {
    printf(
      "Species number %d is %s\n",
      *p_count++,
      Libnucnet__Species__getName( p_species )
    );
  }

  return 1;

}
```

Then to apply this routine, the user calls it from his or her program:

```
i_count = 0;
Libnucnet__Nuc__iterateSpecies(
  p_my_nuclei,
  (Libnucnet__Species__iterateFunction) my_counter_and_printer,
  &i_count
);
```

In this example, the code initializes i_count to zero and then iterates over the species included in p_my_nuclei and applies my_counter_and_printer to each species. Note that the species are iterated in the order in which they were stored [or were previously sorted, if the user previously called Libnucnet__Nuc__sortSpecies()]. Examples in the libnucnet distribution provide further details and examples on how to write, apply, compile, and link iterators.

3 Iterating over Reactions

To iterate over reactions in a reaction collection, the user supplies a routine to apply to a reaction iteration. The prototype is

```
int
my_reaction_iterate_function(
  Libnucnet__Reaction *p_reaction,
  void *p_my_data
);
```

The routine may have any appropriate name. The necessary inputs are:

- **p_reaction:** A pointer to a reaction.
- **p_my_data:** A pointer to a user-defined data structure containing extra data to be passed to the user's iterate function.

The routine must return 1 to continue iteration or 0 (zero) to stop.

The user then calls the function to apply with the Libnucnet__Reac API routine Libnucnet__Reac__iterateReactions() routine, which has the prototype

```
void
Libnucnet__Reac__iterateReactions(
  Libnucnet__Reac *self,
  Libnucnet__Reaction__iterateFunction pf_function,
  void *p_user_data
);
```

The necessary inputs are:

- **self:** A pointer to a Libnucnet__Reac structure, which contains the collection of reactions.

- **pf_function:** The name of the user's function to be applied during the iteration. Typically, this needs to be cast as a Libnucnet__Reaction__iterateFunction.

- **p_my_data:** A pointer to a user-define data structure containing extra data to be passed to the user's iterate function. If there are no extra data, this should be NULL.

Again, examples in the libnucnet distribution provide further demonstration of the user of reaction iterators. Note that the reactions are iterated in alphabetical order according to the reaction string.

4 Iterating over Zones

Iterating on zones is analogous to iterating on nuclides or reactions. The iterate function has the prototype

```
int
my_iterate_function(
  Libnucnet__Zone *p_zone,
  void *p_my_data
);
```

The routine may have any appropriate name. The necessary inputs are:

- **p_zone:** A pointer to a zone.

- **p_my_data:** A pointer to a user-defined data structure containing extra data to be passed to the user's iterate function.

The routine must return 1 to continue iteration or 0 (zero) to stop.

To iterate over the zones, the user then calls Libnucnet__iterateZones(), which has the prototype:

```
void
Libnucnet__iterateZones(
  Libnucnet *self,
  (Libnucnet__Zone__iterateFunction) pf_function,
  void *p_user_data
);
```

The necessary inputs are:

- **self:** A pointer to a Libnucnet structure, which contains the collection of zones.

- **pf_function:** The name of the user's function to be applied during the iteration. Typically, this needs to be cast as a Libnucnet__Zone__iterateFunction.

- **p_my_data:** A pointer to a user-define data structure containing extra data to be passed to the user's iterate function. If there are no extra data, this should be NULL.

Zones are iterated in alphabetical order by their labels.. Examples in the libnucnet provide further details.