# Webnucleo Technical Report: libnucnet Iterators

Bradley S. Meyer

May 4, 2010

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. As of version 0.3, it is possible to iterate over reaction elements (reactants and products) and over optional properties assigned to a zone.

## 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-defined 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. As of version 0.4, reactions are iterated in the order in which they are stored internally in the Libnucnet__Reac structure. The user does not define this. To iterate in a different order, the user supplies a Libnucnet__Reaction__compare_function and sets it with Libnucnet__Reac__setReactionCompareFunction(). To restore the default, the user should call Libnucnet__Reac__clearReactionCompareFunction(). If the number of reactions is large, the iteration in the default order can be much faster than one that requires a sorting. For this reason, the default should be used where possible.

# 4 Iterating over Reaction Elements

Reaction elements are reactants or products in a reaction. As of version 0.3, it is possible to iterate over them. To do so, the user supplies a routine to apply to a reaction element iteration. The prototype is

```
int
my_reaction_element_iterate_function(
  Libnucnet__Reaction__Element *p_element,
  void *p_my_data
);
```

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

- **p_element:** A pointer to a reaction element (a reactant or product).
- **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__Reaction__iterateReactants() or Libnucnet__Reaction__iterateProducts() routine, which have the prototypes

```
void
Libnucnet__Reaction__iterateReactants(
  Libnucnet__Reaction *self,
  Libnucnet__Reaction__Element__iterateFunction pf_function,
  void *p_user_data
);
```

and

```
void
Libnucnet__Reaction__iterateProducts(
  Libnucnet__Reaction *self,
  Libnucnet__Reaction__Element__iterateFunction pf_function,
  void *p_user_data
);
```

The necessary inputs for both are:

- **self:** A pointer to a Libnucnet__Reaction structure, which contains the collection of reactants and products.
- **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__Element__iterateFunction.
- **p_my_data:** A pointer to a user-defined 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 element iterators. Note that the reaction elements are iterated in the order in which they were stored. The Libnucnet__Reac API routine Libnucnet__Reaction__Element__isNuclide() is convenient for distinguishing between nuclide and non-nuclide reactants or products while the routine Libnucnet__Reaction__Element__getName() retrieves the reaction element name.

# 5 Iterating over Zones

Iterating on zones is analogous to iterating on nuclides, reactions, or reaction elements. 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_user_data:** A pointer to a user-defined data structure containing extra data to be passed to the user's iterate function. If there are no extra data, this should be NULL.

As of version 0.4, zones are iterated in the order in which they are stored internally in the Libnucnet structure. The user does not define this. To iterate in a different order, the user supplies a Libnucnet__Zone__compare_function and sets it with Libnucnet__setZoneCompareFunction(). To restore the default, the user should call Libnucnet__clearZoneCompareFunction(). If the number of zones is large, the iteration in the default order can be much faster than one that requires a sorting. For this reason, the default should be used where possible.

# 6 Iterating over Zone Optional Properties

As of version 0.3, it is possible to assign optional properties to a zone. An optional property is stored as a string and is identified by another string giving its name and up to two optional tags. It is likely a user will retrieve the property by an API routine; nevertheless it is the case that one may want to iterate over the optional properties of a zone. This is possible. The iterate function to be applied during such an iteration has the prototype

```
int
my_iterate_function(
```

```
  const char *s_name,
  const char *s_tag1,
  const char *s_tag2,
  const char *s_value,
  void *p_user_data
);
```

The routine may have any appropriate name. The routine must return 1 to continue iteration or 0 (zero) to stop. The necessary inputs are:

- **s_name:** A string giving the name of the property.
- **s_tag1:** A string giving the first tag of the property.
- **s_tag2:** A string giving the second tag of the property.
- **s_value:** A string giving the value of the property.
- **p_user_data:** A pointer to a user-defined data structure containing extra data to be passed to the user's iterate function.

To iterate over the zone's optional properties, the user then calls Libnucnet__iterateZones(), which has the prototype:

```
void
Libnucnet__Zone__iterateOptionalProperties(
  Libnucnet__Zone *self,
  const char *s_name,
  const char *s_tag1,
  const char *s_tag2,
  Libnucnet__Zone__optional_property_iterate_function pf_function,
  void *p_user_data
);
```

The necessary inputs are:

- **self:** A pointer to a Libnucnet__Zone.
- **s_name:** The property name. NULL is considered to match on all property names. If a name is supplied, the iteration will be over properties that match that name.
- **s_tag1:** The first tag. NULL is considered to match on all first tags. If tag1 is supplied, the iteration will be over properties that match that tag.
- **s_tag2:** The second tag. NULL is considered to match on all second tags. If tag2 is supplied, the iteration will be over properties that match that tag.

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

As of version 0.6, properties are iterated in the order in which they are stored internally. The user does not have control over this ordering. This is a change from previous versions in which the iteration was alphabetical over the properties according to their name or tag. The new iteration scheme can be considerably faster for a large number of properties. Examples in the libnucnet distribution provide further details.