Requirements and specifications for a particle database

WPEC Subgroup 38

May 5, 2015

Contents

1 Introduction

One of the tasks of WPEC Subgroup 38 (SG38) is to design a database structure for storing the particle information needed for nuclear reaction databases and transport codes. Since the same particle may appear many times in a reaction database (produced by many different reactions on different targets), one of the long-term goals for SG38 is to move towards a central database of particle information to reduce redundancy and ensure consistency among evaluations. The database structure must be general enough to describe all relevant particles and their properties, including mass, charge, spin and parity, half-life, decay properties, and so on. Furthermore, it must be broad enough to handle not only excited nuclear states but also excited *atomic* states that can de-excite through atomic relaxation. Databases built with this hierarchy will serve as central repositories for particle information that can be linked to from codes and other databases. It is hoped that the final product is general enough for use in other projects besides SG38.

While this is called a 'particle database', the definition of a particle (as described in Section 2) is very broad. The database must describe nucleons, nuclei, excited nuclear states (and possibly atomic states) in addition to fundamental particles like photons, electrons, muons, etc. Under this definition the list of possible particles becomes quite large. To help organize them the database will need a way of grouping related particles (e.g., all the isotopes of an element, or all the excited levels of an isotope) together into particle 'groups'. The database will also need a way to classify particles that belong to the same 'family' (such as 'leptons', 'baryons', etc.). Each family of particles may have special requirements as to what properties are required.

One important function of the particle database will be to provide an easy way for codes and external databases to look up any particle stored inside. In order to make access as simple as possible, the database will include unique names (or 'id's) for every particle that it stores. Reaction databases can then refer to a specific particle either by id. Where possible, particle ids should be made descriptive to assist human readers in understanding the contents of reaction and particle databases.

The particle database is being primarily developed to store nuclei and nuclear states, but it must also be capable of storing some atomic and molecular properties. In particular the database must support atomic electron configurations since they can play an important role in nuclear reactions and decays. Excited nuclear states sometimes decay via internal conversion, 'kicking out' an electron from an inner shell and leaving the remaining electrons to de-excite (emitting x-rays) to fill the new vacancy. Excited atomic states can also be populated through photo-atomic reactions Adding electronic configurations to the database has the potential to drastically increase its size: if unique ids must be given not only to each nucleus and excited nuclear level but also to every possible electron configuration for each nucleus and level, the database will quickly grow to an unmaintainable size. In order to get around this problem and support atomic and molecular properties without drastically increasing the number of particles, the database will allow the use of 'qualifiers' that can be added to a particle to modify its properties. For example, the qualifier 'electronVacancy="1s1/2"' could be added to an isotope or excited nuclear level to indicate a vacancy in one of the electron shells. Qualifiers will be discussed in more detail in section 18.5.

This document describes the requirements and specifications for a particle data hierarchy, including documentation, bibliography information, particle qualifiers as well as particle families and particle groups. This document uses the XML meta-language to illustrate examples of how particle data will be stored. However, like other tasks under SG38, it should be possible to store the particle database in other meta-languages (that is, languages that define a general syntax that can be used to define more specific languages) that support nested hierarchies.

2 Definitions

The following terms will be used frequently throughout this document:

- **matter**: Any small, localized object such as an atom, isotope or elementary particle. Every 'matter' instance in the particle database is assigned a unique id.
- **Particle**: A small, localized object that can be attributed properties such as mass, charge, spin, parity, half-life and decay properties. This definition is deliberately broad to include nuclei and excited nuclear states as well as fundamental particles like the electron and photon.
- **Particle Family**: A set of particles that can be classified together (examples include leptons, baryons, isotopes and excited nuclear levels). Particle families may add additional properties that must be defined for all their particles (i.e. lepton number for all leptons, and level energy for all excited nuclear levels).
- **Particle Group**: A particle group is a set containing related particles so that they all appear together within a particle database hierarchy. This is useful when a set of particles have properties in common: the common properties are defined in the particle group, and inherited by all particles in that group. The excited levels of an isotope are an example: they share properties like the proton count Z and the mass number A.
- **id**: A string used to identify and refer to a particle. Each particle in the database must have a unique id.

- **qualifier**: Extra key that may be appended to a particle id to give additional information about a specific instance of that particle.

3 Requirements

In this section, the key words ‘shall’ and ‘should’ will be used to differentiate between requirements and recommendations.

1. Each ‘particle’ and ‘particleGroup’ in the database shall have a unique id used to identify and refer to it. Only these objects shall have ids (for example, no id is given to a mass or spin).

2. A naming convention for ids shall be defined. Creating and adhering to a naming convention makes comparing and merging different particle databases simpler. If a user wishes to refer to a particle by a name that is not consistent with the naming scheme, they can still do so by defining an ‘alias’ for that particle (see requirement 10).

3. Every particle shall contain at least the following properties: mass, charge, spin, parity and half-life. However, some of these properties may be inherited from higher in the hierarchy rather than being listed explicitly (see requirement 5).

4. The database shall support storing uncertainties with each particle property. Multiple types of uncertainty should be supported, including central values with uncertainty (for example, mass = 54.938 $\pm$ 0.729 amu), upper/lower uncertainty intervals (e.g., mass = 0.03 eV/c^2 - 0.02 + 0.06), and lists of multiple possible assignments (e.g., spin = 3/2, 5/2 or 7/2). If multiple assignments are listed, the database shall assign one as the ‘recommended’ value.

5. The database shall use nesting and inheritance where possible to reduce redundancy by grouping similar particles together. For example, the database should support grouping isotopes with the same atomic number together inside an element, such that all isotopes inherit the same atomic number Z from the element. It should also support grouping excited nuclear states of the same isotope together, such that the mass of each excited state can be computed from the ground-state mass and excitation energy.

6. The database shall support defining ‘families’ to classify similar particles. Each particle family may have additional required data elements (beyond the list in requirement 3). For example, a ‘nuclearLevel’ family should be defined, where each nuclear level requires a level energy and level index in addition to charge, spin, parity, etc.

7. The database shall support storing decay properties for unstable particles. Decays are organized into decay channels, which consist of a probability and a list of products.

Each product may also have an associated uncertainty. For example, if an excited nuclear level can γ-decay to two different levels, the decay can be stored either as two different decay channels that each contain a probability and one gamma product, or as a single decay channel with two gamma products and their probabilities.

8. The database shall support storing documentation sections at multiple levels, down to the level of an individual particle property like mass or spin.

9. The database shall support a bibliography section. Each item in the bibliography shall include a unique citation label that can be used to refer to it from any documentation section.

10. The database shall support a mechanism for defining aliases for particles. For example, the id "Am242_m1" could be an alias for "Am242_e2". Alias definitions shall be permitted at various places within the particle database.

11. When linking to a particle database, user codes shall be permitted to add extra information about a particle by using 'qualifier' keys. For example, a qualifier key may be defined to describe the electron configuration of an atom following a photo-atomic reaction.

3.1 Discussion

- Question: should there be a pre-defined list of the units that are allowed within the particle database, or should choice of units be up to the user? If we decide on the first, that should be added as another requirement.

- Should the database support flags like 'firm', 'tentative', etc. to indicate confidence in an assignment? Those could be stored in documentation sections, but if user codes need to know which assignments are tentative they shouldn't need to parse documentation to find out.

- Regarding requirement number 7: There are several ways of handling decay radiation, so this requirement needs a bit more discussion.

  In GND, both reaction data and decay data are sorted into channels, where every channel has a different list of outgoing products. If that example is followed in the particle database, then decays to different excited states in the daughter nucleus would each have their own decay channel. Decay spectra are then reconstructed by looking up each daughter product, checking their decay properties, and summing over all products. For example, a nuclear excited state that can decay via two different gammas would be stored as:

```
<nuclearLevel id="C12_e8" levelIndex="8">
```

```
    ...
    <decays>
      <decay mode="gamma" probability="0.87">
        <product pid="gamma"/>
        <product pid="C12_e0"/></decay>
      <decay mode="gamma" probability="0.13">
        <product pid="gamma"/>
        <product pid="C12_e1"/></decay>
    </decays></nuclearLevel>
```

The second decay mode leaves C12 in an excited state. Decay properties of that state (C12_e1) can be found by looking it up in the database. That particle would contain information about another gamma decay that leads to the ground state and emits a 4.4389 MeV gamma. Gamma energies do not need to be explicitly stored, since they can be calculated from the initial and final level energies. This organization is useful for presenting detailed decay spectra that may proceed through multiple paths. It is less useful when the details are uncertain, such as when a decay passes through unknown short-lived intermediate states on its way to a longer-lived product.

Other databases follow a different organization. For example, the ENDF decay sub-library starts by listing the possible decay modes (as in '$\beta+$', 'electron capture', etc.), and then lists the decay radiation (including both discrete and continuous spectra) emitted via all decay modes, potentially including radiation emitted not by the parent but by a short-lived intermediate state. In this case, the C12_e8 decay scheme would become:

```
  <nuclearLevel id="C12_e8" levelIndex="8">
    ...
    <decays>
      <decay mode="gamma" probability="1">
        <product pid="gamma" probability="0.87">
          <energy value="12.71" unit="MeV"/></product>
        <product pid="gamma" probability="0.13">
          <energy value="8.271" unit="MeV"/></product>
        <product pid="gamma" probability="0.13">
          <energy value="4.4389" unit="MeV"/></product>
        <product pid="C12_e0" probability="1.0"/>
      </decay></decays></nuclearLevel>
```

This is a suitable way of storing experimental decay spectra, since experiments are actually measuring the transitions rather than the particles that emitted those transitions. It is also useful when searching through measured decay radiation to find what parent likely emitted the radiation. On the other hand, it obscures the fact that the 8.271 MeV and 4.4389 MeV gammas occur in coincidence. This style also leads to possible redundancy and discrepancies if more than one parent particle can decay through the same intermediate states, since the same gamma products may also be emitted as part of the decay cascade from a higher level.

- There are some special cases that also need to be considered. For example, how should correlated decays be handled? Example: angular correlations between Ni60 gammas following Co60 beta decay. These angular correlations show up because the two M2 gamma transitions happen back-to-back, meaning that the second decay is not truly independent of the first. Is it sufficient to store the fact that each transition is M2, and require user codes to calculate the angular correlations if needed? Or, do we need some explicit markup indicating that these decays are correlated?

- Another question: current decay databases use the internal conversion coefficient (ICC) when a decay can proceed either through gamma emission or through internal conversion. Should we continue to support that convention (grouping all 'electromagnetic' decays together and then using a coefficient to denote relative weights), or should we break them out into separate decay channels?

  Another potentially useful convention is to store branching ratios instead of probabilities: each decay mode has a branching ratio BR = (probability of this decay) / (probability of most likely decay). Should we continue to support that convention, or always just store the decay probabilities?

- Should the requirements include a way to describe how 'complete' a level scheme is (i.e., how many levels starting from the ground state have firm spin/parity and energy assignments)? Also, should there be a way of specifying rotational bands for excited nuclear levels?

4 Specifications overview

The rest of this paper describes the structure of the particle database hierarchy, along with XML examples to clarify the specifications. The paper is broken up into sections corresponding to the components of the particle database. Each section starts with a brief discussion, followed by a list of proposed specifications for storing and using that type of data.

The following sections present proposed particle naming conventions, followed by a proposal for a common container for storing physical quantities (including uncertainties).

Specifications are then given for the full particle hierarchy, starting with the top-level element (the ‘particleDatabase’), and descending down. For each element in the hierarchy, the required and optional attributes and sub-elements are listed along with a general description of the type of data stored at that level. Where applicable, design choices in the hierarchy will be tied to specific requirements from section 3.

5 Particle naming schemes

Since particle ids are used to access particles from codes or from other databases, they shall follow a naming convention in order to be easy to recognize and identify across multiple particle databases. The naming scheme should be consistent, applying the same rules to each particle family where possible. It should be designed to give short, easily-recognized names to the most commonly used particles.

While the naming scheme is meant to be strictly adhered to, users will also be able to define aliases in case they prefer a different naming scheme. The meanings of some common aliases are also specified, to ensure that they point to the expected particle.

5.1 Specifications

- Ids for common particles include ‘e-’ for the electron, ‘n’ and ‘p’ for the neutron and proton, ‘photon’ for the photon and ‘nu_e-’ for the electron neutrino.
- Ids for anti-particles follow the convention ‘particleName_anti’. For example, the positron (anti-particle of the electron) is denoted ‘e-_anti’, although it may also be aliased to ‘e+’.
- Nuclear states including the ground state and excited levels are identified by their atomic symbol, the total number of nucleons ‘A’, and the energy level index stored as ‘_e#’ (where ‘#’ is an integer string equal to the level index). For example, the ground state of the isotope with 26 protons and 30 neutrons has the id ‘Fe56_e0’. The third excited state of the same isotope has the id ‘Fe56_e3’.
- Some common aliases are defined, along with the particle they link to.
  - An isotope name shall be an alias pointing to the ground state. For example, the alias ‘Fe56’ points to the nuclear level ‘Fe56_e0’.
  - Nuclear metastable states are aliases pointing to the appropriate nuclear level. For example, the first metastable state of Am242 is stored with the alias ‘Am242_m1’, which points to the particle with id ‘Am242_e2’. The suffix ‘_m#’ shall only be used to denote metastable states.
  - The alias ‘e+’ is reserved for the positron (anti-particle to the electron). It can be used instead of the particle id ‘e-_anti’.

– Other common aliases?

These proposed naming conventions will be used through the rest of this document.

6 Physical quantities and uncertainty

Particle properties like mass, half-life and charge are all quantities with a value, unit and possible uncertainty information. Simple quantities can be compactly expressed as a string with units, as in 'mass="1.0089 amu +/- 1.1e-3"', where the uncertainty represents 1σ of a normal distribution. Other quantities cannot be expressed so compactly: their uncertainty distribution may not be Gaussian, may be asymmetric, or may be expressed as multiple different confidence intervals. In order to handle all these possibilities, the particle database uses a more flexible (although more verbose) representation for physical quantities and their uncertainties, as described below.

6.1 quantity

The **quantity** is an element storing a single physical value along with a label, unit, and optional documentation and uncertainty information.

6.1.1 Specifications

Tagname: 'quantity'

Attributes:

- **label (required):** string identifying this quantity.
- **value (required):** value of the quantity. The value may be numeric (integer or floating-point), or it may be a string.
- **unit (optional, default=''):** string name of the unit for this quantity. The unit can be omitted if the quantity is unit-less.
- **for (optional, default=''):** link pointing to another quantity that should be associated with this one. See example in section 6.5.
- **confidence (optional, default=''):** Needs detail

Child elements:

- **documentation (optional):** Contains documentation specific to this quantity (how it was derived, etc.).
- **uncertainty (0 or more):** See next section for a description of the 'uncertainty' element.

6.2 uncertainty

A quantity may contain more than one **uncertainty** element. Each one stores a description of an uncertainty distribution or confidence interval. For a symmetric uncertainty distribution, only one 'uncertainty' element is needed. However, multiple **uncertainty** elements are required for more complicated uncertainties such as asymmetric distributions or multiple confidence intervals.

6.2.1 Specifications

Tagname: 'uncertainty'

Attributes:

value (required) : numeric value of this uncertainty

unit (optional, default='sameAsParent') : string containing the unit. Possible values of the 'unit' are:

- 'sameAsParent': indicates an absolute uncertainty, given in the same units as the parent quantity. Thus, if the unit for the parent quantity is 's', 'value="0.1"' indicates an uncertainty of 0.1s.
- 'relative': indicates that the value is a relative uncertainty. Here, 'value="0.1"' indicates an uncertainty of 10%
- '%': indicates a percent uncertainty. Here, 'value="0.1"' indicates an uncertainty of 0.1%
- any other unit: indicates an absolute uncertainty given in the specified unit. The specified unit must be compatible with the unit of the parent quantity.

type (optional, default='variance') : indicates how this uncertainty is to be used. Possible values of 'type' are:

- variance: the value represents 1σ, symmetric around the central value,
- variance-: the value represents 1σ below the central value,
- variance+: the value represents 1σ above the central value, or
- confidence-interval-: the value corresponds to a confidence interval below the central value. This option must be used along with a number in the *pdf* attribute.
- confidence-interval+: the value corresponds to a confidence interval above the central value. This option must be used along with a number in the *pdf* attribute.

pdf (optional, default='normal') : describes the shape of the probability distribution. Possible values of 'pdf' are:

- normal: standard Gaussian distribution, log-normal: log-normal distribution, or

any number in the range [0,1]: indicates a given confidence interval (only for use when ‘type’ is equal to ‘confidence-interval-’ or ‘confidence-interval+’).

Child elements: None

6.3 Particle property

According to requirement 4, the particle database must support storing multiple possible assignments for a quantity. For example, an evaluator may not be able to firmly establish the spin and parity of a particle, but may be able to narrow the list down to a few possibilities. Each of these possibilities is stored in a **quantity** element, and they are grouped together inside a **particle property**.

6.3.1 Specifications

Tagname: determined by the property type: ‘mass’, ‘charge’, ‘spin’, ‘parity’, ‘halflife’, etc.

Attributes:

recommended (required if the set contains more than 1 quantity): xPath link indicating which quantity is considered the best value. The link uses the *label* attribute to uniquely specify one quantity.

Child elements:

quantity (1 or more): The list of possible quantities in this set. Each quantity in the set shall have a unique *label* attribute.

6.4 Examples

If the recommended mass for a paquantityrticle is the atomic mass (“1.0089 amu” with uncertainty “± 1.1e-3 amu”), it can be stored in XML as:

```
<mass recommended=‘quantity[@label="atomic"]’>
  <quantity label="atomic" value="1.0089" unit="amu">
    <documentation>Atomic mass from Audi&Wapstra ...</documentation>
    <uncertainty pdf="normal" type="variance" "value="1.1e-3"/>
  </quantity>
</mass>
```

In the **mass** element, xLink/xPath syntax [1] is used to indicate the recommended value. In this case the *recommended* attribute could be omitted since **mass** contains only one quantity. The *pdf* and *type* attributes on the **uncertainty** element could also be omitted since they are equal to the defaults.

If the same mass value were given with asymmetric uncertainties, it could be stored as:

```
<mass recommended='quantity[@label="atomic"]'>
  <quantity label="atomic" value="1.0089" unit="amu">
    <documentation>Atomic mass from Audi&Wapstra ...</documentation>
    <uncertainty pdf="normal" type="variance-" "value="9e-4"/>
    <uncertainty pdf="normal" type="variance+" "value="1.2e-3"/>
  </quantity>
</mass>
```

The particle structure supports listing multiple possible assignments for a particle's property. In this case the property contains multiple quantities, along with a 'recommended' flag indicating which to use by default. In the example below, the recommended spin is '1/2', but other possible assignments include '3/2' and '5/2':

```
  <spin recommended='quantity[@label="0"]'>
    <quantity label="0" value="1/2" unit="hbar"/>
    <quantity label="1" value="3/2" unit="hbar"/>
    <quantity label="2" value="5/2" unit="hbar"/>
  </spin>
```

The database also supports listing multiple different representations of a quantity type. For example, while the mass of an isotope is typically given in atomic mass units, it may also be given as a mass excess or as binding energy per nucleon. In this case, each representation may have its own associated uncertainty:

```
  <mass recommended='quantity[@label="atomic"]'>
    <quantity label="atomic" value="54.93804514" unit="amu">
      <uncertainty value="7.3e-7"/></quantity>
    <quantity label="massExcess" value="-57710.58" unit="keV">
      <uncertainty value="0.68"/></quantity>
    <quantity label="bindingEnergyPerNucleon" value="8764.988" unit="keV">
      <uncertainty  value="1.2e-2"/></quantity>
  </mass>
```

In both examples, the *recommended* attribute is required to specify which child quantity to use by default. The **uncertainty** elements do not need to specify their *pdf* and *type* since they are both equal to the default values ("normal"' and "variance" respectively).

6.5 Discussion

- In the spin example above, the spin is given as a ratio '1/2' rather than a floating-point equivalent. Another possibility would be to use units of $\hbar/2$ and store all spins as integers, but this may be less familiar and less intuitive to nuclear and particle physicists.

- The examples above do not deal with the possibility (raised earlier in the discussion of requirements) that we may need to support multiple *correlated* assignments, as in: 'spin/parity = 1/2- or 3/2+'. This appears occasionally in RIPL and ENSDF, for example the 3.13 MeV excited state in Mn51 which has spin/parity = either 3/2- or 5/2+.

  One suggestion for handling this is to add a *for* attribute to a **quantity** element, that links two correlated values. For example,

```
<particle id="...">
  ...
  <spin recommended='quantity[@label="0"]'>
    <quantity label="0" value="1/2" unit="hbar"
     for='../../parity[@label="0"]'/>
    <quantity label="1" value="3/2" unit="hbar"
     for='../../parity[@label="1"]'/>
  </spin>
  <parity recommended='quantity[@label="0"]'>
    <quantity label="0" value="-1" unit=""
     for='../../spin/quantity[@label="0"]'/>
    <quantity label="1" value="+1" unit=""
     for='../../spin/quantity[@label="1"]'/>
  </parity>
</particle>
```

  This approach uses explicit links to indicate which spin and parity are meant to be used together. It opens up a possibility for discrepancies, however, if the *for* attribute for a spin points to a parity that does not point back to the same spin.

  One option to resolve that problem would be to only use the *for* attribute to link from the spin to the associated parity, and not vice-versa. In that case, there would be at most two possible parity assignments, and multiple spins could point to the same parity.

  TBD: need an example showing the use of *confidence* attribute for quantities.

7 particleDatabase

The previous section described the **quantity** and **uncertainty** elements that are used to store basic physical data. In the next section we turn to the overall organization of the database, which contains all particles as well as documentation, a bibliography and a list of aliases.

7.1 Specifications

Tag: 'particleDatabase'

Attributes:

formatVersion (required): the format version for the particle hierarchy, required since the format may evolve in the future.

library (required): name of the evaluated particle library.

libraryVersion (required): version of the library.

date (required): The compilation date for this particle library, stored in the format: 'YYYY-MM-DD'.

Child elements:

documentation (required) : see section 8.

bibliography (optional) : see section 9.

aliases (optional) : see section 11.

particle (0 or more) : a list of **particle** elements. The **particle** is described in section 12.

particleGroup (0 or more) : a list of **particleGroup** elements. The **particleGroup** is described in section 18.

7.2 Example

```
<particleDatabase formatVersion="1.0" library="newParticleLibrary"
            libraryVersion="0.1" date="2015_04_29">
  <documentation> ... </documentation>
  <bibliography> ... </bibliography>
  <aliases> ... </aliases>
  <!-- list of particles and particle groups: -->
  <boson> ... </boson>
  ...
  <lepton> ... </lepton>
```

```
    ...
    <chemicalElement> ... </chemicalElement>
    ...
  </particleDatabase>
```

8 documentation

The **documentation** element can appear at several different places in the hierarchy, including inside the **particleDatabase**. The documentation section will follow the same specifications as defined for 'text' containers in the general-purpose data containers requirements document.

8.1 Specifications

Tag: 'documentation'

Attributes:

> **markup:** Markup language used to store this documentation. Possible values include 'xhtml', 'xml' or 'latex'.
>
> **characterSet:** Restricts the allowed characters for use in documentation. Default = 'utf-8', another possible value is 'ascii'.

Contents: Documentation data stored in the indicated style. If the style is 'xhtml' or 'xml', the documentation shall be wrapped inside an XML CDATA section (when stored in XML) to show that it is not meant to be handled by the XML parser.

8.2 Discussion

Even in an xml document, the CDATA may not be necessary so long as the contents are well-formed xml or xhtml. There would be a problem if the section contained reserved xml characters like '<', '>' or '&'.

9 bibliography

The **bibliography** element contains a list of references, each with a unique 'refId'.

XML-based markup for bibliographies already exist. One option is BibTeXML (`http://bibtexml.sourceforge.net/`), an XML representation of the BibTex bibliographic system used by LaTeX. Another option is the openXML schema for bibliographies (`http://www.schemacentral.com/sc/ooxml/s-shared-bibliography.xsd.html`). Further research is needed before deciding whether to use one of these existing schemas or to develop a new version for the particle database. Whichever solution is chosen, it should support storing (at least) a list of references, each with a unique label, title, authors, publication details, url or doi, and some descriptive comments.

9.1 Discussion

- Specifications for the bibliography will probably be adopted from one of the projects mentioned above, so they are not spelled out here.
- Currently the bibliography can appear at the **particleDatabase** level, within a particle group, and/or within a specific particle instance. Should it be permitted anywhere else in the hierarchy?

10 alias

Particles may sometimes be known by more than one name. Examples include the α particle, which is synonymous with the bare He4 nucleus, and isomeric nuclear states which can be referred to either by isomer index or by nuclear level index. Rather than requiring that these particles be listed multiple times (leading to redundancy and possible discrepancies), the particle database shall avoid redundancy by supporting aliases. Each alias consists of a unique id, a link to another particle or alias, and optional additional information.

10.1 Specifications

Tag: 'alias'

Attributes:

id (required): string identifying this alias. The id for every alias (as well as for every particle) must be unique within the database.

pid (required): string containing the id of the particle that this alias refers to.

other attributes? other optional information may be needed to describe the alias. These could include descriptive text, a 'metastable index', ...

Child elements: None

11 aliases

alias elements can be defined at several different places within the particle database. One place is at the top level, inside the **aliases** list.

11.1 Specifications

Tag: ‘aliases’

Attributes: None

Child elements: contains a list of 0 or more **alias** elements

11.2 Example

```
<aliases>
  <alias id="H1" pid="p"/>
  ...
  <alias id="Am242_m1" pid="Am242_e2" metaStableIndex="1"/>
</aliases>
```

11.3 Discussion:

Do all aliases need to be defined at the same place, or should we allow defining a particle and its alias(es) at the same place? For example:

```
<nuclearLevel id="Am242_e2" index="2" alias="Am242_m1">...</nuclearLevel>
```

One problem with this approach: if a single particle has multiple aliases, only one could be defined this way. Perhaps each particle should allow an optional **aliases** sub-element?

See also the discussion on anti-particles in section 17.

12 particle

The remainder of the particleDatabase consists of a list of particles and/or particle groups. This section defines the general features that are common to all particles. The **particle** element can be thought of as the ‘base class’ for storing a particle, from which all other particle families are derived.

12.1 Specifications

Tag: family name (see section 16).

Attributes:

id: String identifying this particle. Must be unique throughout the entire database (including particles and aliases). Suggestion: restrict ids to ascii character set?

name (optional): In addition to the id (which is meant to be descriptive but short), the particle may have a more descriptive full name.

date (optional): Can be used to indicate when the particle was added to a database, or the last time it was modified. Suggestion: make the date required?

Child elements:

documentation (optional): contains documentation specific to this particle. Identical to the particleDatabase/documentation element described in section 8.

bibliography (optional): contains references specific to this particle, identical to the particleDatabase/bibliography element described in section 9.

The following child elements are all required unless inherited from higher up in a particle group (see section 18):

charge: The charge, stored as a quantity group (i.e., **charge** with one or more **quantity** elements inside it as described in section 6).

spin: quantity group.

parity: quantity group

mass: quantity group

halflife: quantity group. Quantities may be stored with units of either time or energy, as in 'halflife="1 fs"' or 'halflife="3.3 GeV"'.

decays (optional): decay information if applicable. See section 15 for details.

12.2 Example

```
<boson id="photon">
  <documentation style="ascii">The photon, first proposed by ...
        </documentation>
  <charge>
    <quantity label="0" value="0" unit="e"/></charge>
  <spin>
    <quantity label="0" value="1" unit="hbar"/></spin>
  <parity>
    <quantity label="0" value="1" unit=""/></parity>
  <mass>
    <quantity label="0" value="0" unit="eV/c**2"/></mass>
  <halflife>
    <quantity label="0" value="stable" unit="s"/></haflife>
```

```
</boson>
```

13 decay

The **decay** element represents a single decay mode, including type of decay, probability and a list of products.

13.1 Specifications

Tag: ‘decay’

Attributes:

- **label (required):** Unique label, used when linking to a specific decay mode.
- **mode (required):** String identifying the decay mode. Some possible values include ‘beta-’, ‘beta+’, ‘alpha’, ‘electroMagnetic’, and ‘spontaneousFission’.

Child elements:

- **probability (required):** Decay probability, stored as a particle property (see section 6.3). The probability for all decays shall sum to 1.0, to within a tolerance defined by library managers.
- **Q (optional):** Decay Q-value, stored as a particle property.
- **ICC (optional):** Internal conversion coefficient, stored as a particle property. Meant for use with ‘electroMagnetic’ decays only.
- **product (1 or more):** a list of **product** elements, as described in the next section.

14 product

Each decay product is listed individually as a **product** element.

14.1 Specifications

Tag: ‘product’

Attributes:

- **pid (required)** : a string with the id of the product particle.

Child elements:

- **energy (optional):** Outgoing energy for this product, stored as a set of quantities.
- **probability (optional):** Probability that this product is emitted by the decay.

14.2 Discussion

The optional **probability** element within a product is meant for situations where many decay spectra are confusing, and rather than breaking up into multiple different **decay** elements the evaluator prefers to give a set of products with probabilities.

15 decays

Decay and product data for each product are are organized inside a **decays** element:

15.1 Specifications

Tag: ‘decays’

Attributes: none

Child elements: list of **decay** elements, as described in section 13.

15.2 Examples

```
<baryon id="n" name="neutron">
  ...
  <halflife>
    <quantity label="0" value="613.9" unit="s">
      <uncertainty value="0.6"/></quantity>
  </halflife>
  <decays>
    <decay mode="beta-">
      <probability>
        <quantity value="1"/></probability>
      <Q>
        <quantity label="0" value="782.347" unit="keV">
          <uncertainty value="1e-3"/></quantity>
      </Q>
      <product pid="p"/>
      <product pid="e-"/>
      <product pid="nu_e-_anti"/>
    </decay>
  </decays>
</baryon>
```

15.3 Discussion

- Breaking the decays of a particle into multiple independent decay modes is conceptually simple, but it may not be flexible enough for all existing data. The particle database already supports storing gamma-decay and internal conversion decay together (using the ICC to indicate the relative likelihood of the two modes). Other types of combined decay info may also need to be supported, especially if ENSDF-style data are to be stored in the particle database.
- What about storing decays that have a positive Q-value but have not been measured? Should the database support decay probability limits like '<0.01%'? If so, perhaps the probability should be given as a physical quantity with uncertainty?
- Should there be a special container for the **probability**? Currently it is treated like other particle properties as defined in section 6.3. However, unlike the mass or spin, may not want to allow multiple possible assignments for the probability.

16 Particle families

The particleDatabase can store many different particles. These can be divided into families with similar properties. The family of any particle can be determined by the tag name used to store that particle (i.e., 'boson', 'lepton', 'baryon', 'nuclearLevel', etc.). Particles belonging to each family may require special attributes beyond the list described in section 12 to be fully specified.

Particle families currently supported include:

boson: for the photon (also gluon, W and Z if needed)

lepton: e-, muon, tau, neutrinos and their anti-particles. In addition to the basic set of properties, these also require a *generation* attribute: for e- and e+ the generation is '1', for the muon it is '2', etc.

quark: Like leptons, quarks require a *generation* attribute.

baryon: Includes the nucleons 'n' and 'p' as well as their exotic cousins like Δ, Ξ and Ω. In addition to basic properties, baryons require *isospin* and *flavour* attributes and a list of their constituent quarks.

meson: Includes π^0, $K^{\pm}$, η, etc. In addition to basic properties, mesons are classified by their isospin, flavour, C-parity and G-parity attributes, and a list of constituent quarks.

nuclearLevel: Includes all ground and excited states of atomic nuclei. In addition to basic properties, nuclearLevels must store a level index and level energy, and also possibly information on rotational bands. The nuclearLevel is also different from other particle families in that it does *not* contain an explicit mass or charge. Instead, these quantities are inherited from higher up in a particle group, as described in section 18.

17 anti-particles

Particles and their anti-particles are closely related, sharing many properties. The particle database supports defining a particle and its anti-particle at the same time, using an **antiParticle** element inside the particle definition. An anti-particle comes along with a set of rules for converting from the particle to the anti-particle.

17.1 Specifications

Tagname: 'antiParticle'

Attributes:

id: According to the naming convention defined in section 5, the id for an anti-particle shall be the particle id + '_anti'.

alias (optional): Permits defining one alias for the anti-particle.

Child elements: None

17.2 Example

```
<lepton id="e-" name="electron">
  <mass>
    <quantity label="0" value="0.511" unit="MeV/c**2"/></mass>
  <charge>
    <quantity label="0" value="-1" unit="e"/></charge>
  <antiParticle id="e-_anti" alias="e+"/>
  ...</lepton>
```

17.3 Discussion

Codes reading this in would be responsible for creating two particles ('e-' and 'e-_anti') as well as an alias ('e+', pointing to 'e-_anti'). The two particles have all properties in common except for their ids, lepton numbers and charge. The relationship between a particle and its anti-particle depends on the particle family. For a lepton, converting from particle to anti-particle means multiplying both the charge and lepton number by -1. Where should these relationships be defined? Should they be part of this specifications document?

18 Particle groups

In the particleDatabase, each excited nuclear state is considered to be a unique particle. However, these nuclear excited state particles have many common properties, and can be grouped together under the same isotope. Furthermore, isotopes can also be grouped together by their nuclear charge Z into chemical elements. Grouping similar particles helps reduce redundancy in the particleDatabase, since it means that common attributes like the nuclear charge can be stored in one place and inherited by all particles in the group.

Currently, the only defined particle groups are the **chemicalElement** and **isotope**, although a **rotationalBand** particle group could also be defined to contain nuclearLevels in the same rotational band. These groups are not particles and do not have 'id' attributes. Instead, they are used to organize particles.

This causes trouble if we want **chemicalElement** instances to have an id. That may be justification for keeping the concept of a 'matter' type in the database (in addition to the particle type): matter is not a real particle, but still has an id that we can refer to.

18.1 chemicalElement

18.1.1 Specifications

Tag: 'chemicalElement'

Attributes:

- **name:** full name of the element, as in 'Iron'.
- **symbol:** chemical symbol, as in 'Fe'.
- **Z:** number of protons for this element, stored as an integer.

Child elements:

- **documentation (optional):** documentation common to all isotopes of this element
- **bibliography (optional):** bibliography for all isotopes of this element
- **qualifiers (optional):** See description in section 18.5
- **isotope (0 or more):** described in the next section

18.2 isotope

18.2.1 Specifications

Tag: 'isotope'

Attributes:

name (required): combination of the chemical symbol and the total number of nucleons 'A'. For example, 'Fe56'. See discussion below on the relationship between the isotope name and the ground state nuclearLevel.

A (required): total number of protons + neutrons, stored as an integer.

Child elements:

documentation (optional): documentation common to all nuclearLevels in this isotope.

bibliography (optional): bibliography specific to this isotope

qualifiers (optional): see description in section 18.5.

mass (required): The ground state mass of this isotope, stored as quantity group.

nuclearLevels (required): Contains a list of **nuclearLevel** elements, as described in section 18.3.

18.3 nuclearLevel

Within the isotope, each nuclearLevel is a unique particle. However, the structure of the **nuclearLevel** element is different from that of other particles described in this document, since some data is inherited from higher up in the particle group. Each nuclearLevel is organized as follows:

18.3.1 Specifications

Tag: 'nuclearLevel'

Attributes:

id: the id style is 'SymA_e#', as in 'Mn55_e2' for the excited state of Mn55 with levelIndex="2".

levelIndex: the excited state index stored as an integer.

Child elements: the **nuclearLevel** contains the same elements as other particles, except that it normally does not contain 'mass' or 'charge' particle properties since they are inherited from the parent **isotope**. Also, it contains the following:

energy: the excitation energy for this level, stored as a particle property.

band (optional): identifies the rotational band (if any) that this **nuclearLevel** belongs to.

18.3.2 Discussion

Should the **nuclearLevels** element support flags telling how complete the level information is (similar to RIPL's 'max complete' flag)?

18.4 Examples

```
<chemicalElement name="Manganese" Z="25" symbol="Mn">
  <isotope name="Mn55" A="55">
    <mass recommended='quantity[@label="atomic"]'>
      <quantity label="atomic" value="44.00687" uncertainty="5.4e-4" unit="amu"/>
      <!-- other values of mass if applicable -->
    </mass>
    <nuclearLevels>
      <nuclearLevel id="Mn55_e0" index="0" alias="Mn55">
        <energy>
          <quantity label="0" value="0" unit="eV"/></energy>
        <spin>
          <quantity label="0" value="5/2" unit="hbar"/></spin>
        <parity>
          <quantity label="0" value="-1" unit=""/></parity>
        <halflife>
          <quantity label="0" value="stable" unit="s"/></halflife>
      </nuclearLevel>
      <nuclearLevel id="Mn55_e1" index="1">
        <energy>
          <quantity label="0" value="125.949" unit="keV"/></energy>
        <spin>
          <quantity label="0" value="7/2" unit="hbar"/></spin>
        <parity>
          <quantity label="0" value="-1" unit=""/></parity>
        <halflife>
          <quantity label="0" value="2.59e-10" unit="s"/></halflife>
        <decays>
          <decay index="0" mode="gamma">
            <probability>
              <quantity label="0" value="1.0" unit=""/></probability>
            <product pid="Mn55_e0"/>
            <product pid="gamma"/>
          </decay>
        </decays>
      </nuclearLevel>
      ...
    </nuclearLevels>
  </isotope>
</chemicalElement>
```

Second example:

```
  <chemicalElement name="Magnesium" Z="12" symbol="Mg">
    <isotope name="Mg32" A="32">
      <mass recommended=‘quantity[@label="atomic"]’>
        <quantity label="atomic" value="31.998975" unit="amu">
          <uncertainty value="1.9e-5"/></quantity>
      </mass>
      <nuclearLevels>
        <nuclearLevel id="Mg32_e0" index="0" alias="Mg32">
          <energy>
            <quantity label="0" value="0" unit="eV"/></energy>
          <spin>
            <quantity label="0" value="0" unit="hbar"/></spin>
          <parity>
            <quantity label="0" value="1" unit=""/></parity>
          <halflife>
            <quantity label="0" value="0.095" unit="s"/><halflife>
          <decays>
            <decay index="0" mode="betaMinus">
              <probability>
                <quantity label="0" value="0.97656" unit=""/></probability>
              <Q>
                <quantity label="0" value="10.1" unit="MeV"/></Q>
              <product pid="Al32"/>
              <product pid="e-"/>
              <product pid="nu_e-_anti"/></decay>
            <decay index="1" mode="betaMinusDelayedNeutron">
              <probability>
                <quantity label="0" value="0.02344" unit=""/></probability>
              <product pid="Al31"/>
              <product pid="n"/>
              <product pid="e-"/>
              <product pid="nu_e-_anti"/></decay></decays>
        </nuclearLevel>
        <nuclearLevel id="Mg32_e1" index="1">
          ...
      </nuclearLevels>
    </isotope>
  </chemicalElement>
```

Discussion: • The reader may observe that the **isotope** is not being treated as a particle: it does not have an *id* attribute. However, since the isotope name is often synonymous with the ground state of that isotope, an alias has been entered in the ground state nuclearLevel in both examples, so that 'Mn55' is an alias for 'Mn55_e0' and 'Mg32' is an alias for 'Mg32_e0'.

18.5 qualifiers

Users of particle databases may need information about electron configurations in atoms and molecules. These add a new dimension to the space of particles. While it is feasible to treat every excited nuclear state (for every isotope) as a unique particle with its own id, adding the new dimension of possible electron configurations for each of those states would make the particle database unwieldy. TODO: add brief discussion of size required just to store all these names.

To resolve this problem, the particle database permits storing 'qualifiers' that are used to add extra information to a given particle. The main purpose of qualifiers (at least for now) will be to describe electron configurations and atomic relaxation. Since these properties are element-specific (and in some rare cases isotope-specific), the qualifiers are defined within a 'chemicalElement' and/or 'isotope'.

Each qualifier is a string that can be appended to a particle id inside braces '{' and '}'. More than one qualifier can be stored in the braces as a ';'-delimited list.

Need to flesh out this discussion further. It is likely to change as we start handling more decay and atomic relaxation data. Also, are qualifiers flexible enough to handle simple molecules in addition to excited or ionized atoms? If not, do we need to come up with special types of particles to support $S(\alpha, \beta)$ (thermal scattering) data?

19 Final discussion

- The particle database currently does not contain any information about natural abundances of different chemical elements. While this is important information, it was decided that abundance is not a property of the element, but rather of the location where the element is found: chemical abundances vary at different locations both on earth and throughout the rest of the universe. These data should be stored in an external database instead.

- The particleDatabase can be divided up into multiple files (for example, one file per chemicalElement). Do we need any explicit rules for how to handle that (i.e. require an explicit link to the file containing the subsection)?
- For a particle that is actually made up of smaller constituent particles, should the particle database support storing the list of constituents? For nucleons this is straightforward (a list of up and down quarks with multiplicities), but many other particles are actually a superposition of states rather than a simple composition.

References

[1] XML Linking Language (XLink) Version 1.1. See: http://ww.w3.org/TR/xlink11