Properties of Particles (PoPs) database

WPEC Subgroup 38 semi-annual meeting, 21-22 May 2015

Caleb M Mattoon

Lawrence Livermore National Laboratory

LLNL-PRES-XXXXXX

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

PoPs specifications are similar to version discussed at last SG38, with some recent changes following internal discussion

Changes:

- Created a generic ‘quantity’ container for storing physical quantities
  - Partly to simplify handling multiple related quantities, e.g. atomic mass/nuclear mass/neutron and proton separation energies/mass excess/...
  - also for handling multiple possible assignments, e.g. for spin and parity
- Expanded the ‘uncertainty’ element to allow different types of uncertainty
- Refined ‘qualifiers’ to apply only within a particle group. For example, a qualifier may by defined within a <chemicalElement> and describe electron configurations and decays specific to that element.

Some definitions:

- Particle: A small, localized object that can be attributed properties such as mass, charge, spin, parity, half-life and decay properties.
- Particle Family: A set of particles that can be classified together (examples include leptons, baryons, isotopes and excited nuclear levels).
- Particle Group: A particle group is a set containing related particles (that have properties in common). For example, all the excited levels of an isotope belong together in a particle group.
- id: A string used to identify and refer to a particle. Each particle in the particle 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.

Draft requirements for PoPs:

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

- A naming convention for ids shall be defined.

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

- The database shall support storing uncertainties with all particle properties. Multiple types of uncertainty should be supported, including central values with uncertainty, asymmetric distributions, and lists of multiple possible assignments. If multiple assignments are listed, the database shall require that one assignment be explicitly listed as the ‘recommended’ value.

Draft requirements for PoPs, continued:

- The database shall use nesting and inheritance where possible to reduce redundancy by grouping similar particles together.
- The database shall support defining ‘families’ to classify similar particles. Each particle family may have additional required data elements
- 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.
- The database shall support storing documentation sections at multiple levels, down to the level of an individual particle property like mass or spin.

Draft requirements for PoPs, continued:

- 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.

- The database shall support a mechanism for defining aliases for particles. For example, the id "Am242_m1" could be an alias for "Am242_e2". Aliases may be defined in more than one location in the database.

- 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.

Naming conventions for particle ids. Particle databases should all adhere to these conventions to keep consistency between databases

- ‘e’ for electron, ‘n’ and ‘p’ for neutron and proton, ‘photon’ for all photons, ‘nu_e’ for electron neutrino, ...
- Chemical elements are named by symbol, first letter capitalized: ‘Fe’, ‘U’, etc.
- Isotopes have A appended to the symbol: ‘Fe56’, ‘U235’, etc.
- Excited nuclear states indicated by isotope + ‘_e’ + level index: ‘Fe56_e2’, ‘U235_e0’ (for the ground state)
- Antiparticle indicated by trailing ‘_anti’: ‘p_anti’, ‘e_anti’

Naming conventions also include some aliases:

- Nuclear metastable states are indicated by isotope + '_m' + metastable index. For example, 'Am242_m1' is an alias pointing to 'Am242_e2'
- Several common aliases are reserved to indicate photons: 'x-ray', 'gamma', etc.
- 'e+' is an alias pointing to 'e-_anti'

‘quantity’ and ‘setOfQuantities’ are the low-level containers, used to store mass, charge, spin, parity, energy, halflife, etc.

```
<quantity label="..." value="..." unit="..." kind="..." origin="..." likelihood="..." for="...">
   <documentation .../>
   <uncertainty .../></quantity>
```

- Tag name: ‘quantity’
- Attributes:
  - Required attributes ‘label’, ‘value’ and ‘unit’.
    - ‘label’ is a unique string, used for linking
    - ‘value’ is usually a float, but may be a string like ‘stable’ (for half-life)
    - use unit=“” for dimensionless quantity
  - Optional attributes ‘kind’, ‘origin’, ‘likelihood’ and ‘for’.
    - ‘kind’: examples are ‘atomic’, ‘bindingEnergyPerNucleon’, etc.
    - ‘origin’: how value was determined. ‘measured’, ‘estimated’, etc. Should this be part of documentation instead? Will anyone be using this flag?
    - ‘likelihood’: indicates relative likelihood when multiple assignments are possible
    - ‘for’: link to a different correlated quantity. Discussed further later

‘quantity’ and ‘setOfQuantities’ are the low-level container, used to store mass, charge, spin, parity, energy, halflife, etc.

```
<quantity label="..." value="..." unit="..." kind="..." origin="..." likelihood="..." for="...">
  <documentation .../>
  <uncertainties><uncertainty pdf="..." value="..." unit="..."/></uncertainties></quantity>
```

- Child elements:
  - documentation (optional): lowest level where documentation appears
  - uncertainties (optional): list of uncertainty with:
    - attribute pdf (optional, default=“normal”). Type of probability distribution.
    - Other attributes and/or child elements depend on the value of ‘pdf’.
      - If pdf=“normal”, only need to supply value (i.e., std. deviation) and optionally a unit (if different from the quantity unit)
      - More complex pdfs will also be supported: non-normal distributions, uncertainty intervals, asymmetric uncertainties, etc.
    - Attribute unit (optional, default=“sameAsParent”)

Quantities are stored together inside a set, such as ‘mass’, ‘halflife’, etc. Some examples:

```
<mass>
   <quantity label=“atomic” value=“7.016004548” unit=“amu” >
      <uncertainties><uncertainty value=“8.4e-8”/></uncertainties></quantity>
   <quantity label=“bindingEnergyPerNucleon” value=“…” unit=“…”/>
</mass>

<halflife><quantity label=“0” value=“stable” unit=“s”/></halflife>


<halflife>
     <quantity label=“0” value=“2” unit=“ps” origin=“measured”>
        <uncertainties>
         <uncertainty type=“variance-” value=“1” unit=“ys”/>
         <uncertainty type=“variance+” value=“2”/></uncertainties>
     </quantity></halflife>
```

<uncertainty> is wrapped inside <uncertainties>. More verbose, but helps extend to more complicated uncertainties:

<uncertainty pdf="normal" value="1.23" unit="eV"/>

becomes

<uncertainties><uncertainty pdf="normal" unit="eV"/></uncertainties>

for more complicated case:

```
<uncertainties>
   <uncertainty type="confidenceInterval" pdf=".66" value="…">
   <uncertainty type="confidenceInterval" pdf=".97" value="…">
</uncertainties>
```

or

```
<uncertainties>
   <uncertainty type="variance-" value="…"/>
   <uncertainty type="variance+" value="…"/></uncertainties>
```

If a set of quantities contains more than one quantity, also need a 'recommended' flag to indicate one 'default' value:

```
<mass recommended="atomic">
    <quantity label="atomic" value="7.016004548" unit="amu">
      <uncertainty pdf="normal" value="8.4e-8"/></quantity>
    <quantity label="massExcess" value="14908.141" unit="keV" >
      <uncertainty pdf="normal" value="0.079"/></quantity>
    <quantity label="bindingEnergyPerNucleon" value="5606.291" unit="keV">
      <uncertainty pdf="normal" value="0.011"/></quantity></mass>
```

Should 'recommended' be a link instead of a label?

i.e. <mass recommended='quantity[@label="atomic"]'/>

Spin/parity add another complication: uncertainty here is expressed in alternate possible assignments

- To indicate that spin is most likely 3/2, but could also be 5/2:

Potential confusion here: 'recommended' refers to a label, *not* a spin value. A link (as in 'recommended=quantity[@label="0"]') is more explicit

```
<spin recommended="0">
  <quantity label="0" value="3/2" unit="hbar" origin="estimated" likelihood="high"/>
  <quantity label="1" value="5/2" unit="hbar" origin="alternate" likelihood="low"/>
</spin>
<parity>
   <quantity label="0" value="+" unit=""/></parity>
```

What values are allowed in the 'likelihood' field? Limit to 'high', 'medium', 'low' and numeric values between 0 and 1?

How to store correlated multiple assignments?

- What if evaluator indicates spin/parity is probably 3/2+, but could also be 3/2- or 5/2-?
  - This is where the 'for' attribute comes in:

```
<particle>
  <spin recommended="0">
    <quantity label="0" value="3/2" unit="hbar" kind="spin"
              for='../../parity/quantity[@label="0"]' likelihood="high"/>
    <quantity label="1" value="3/2" unit="hbar" kind="spin"
              for='../../parity/quantity[@label="1"]' likelihood="low"/>
    <quantity label="2" value="5/2" unit="hbar" kind="spin"
              for='../../parity/quantity[@label="1"]' likelihood="low"/>
  </spin>
  <parity recommended="0">...</parity></particle>
```

Top level of the PoPs hierarchy contains documentation, aliases, and qualifiers followed by a list of particles and particle families

- particleDatabase (with formatVersion, library, libraryVersion and date attributes)
  - documentation (identical to GND documentation element)
  - bibliography
    - list of reference elements. Adopt BibTeXML?
  - aliases
    - optional list of alias elements
  - list of particle and/or particleGroup elements.
    - Particle elements may have tags like <nucleon> or <photon> rather than <particle>, and particleGroup elements may be <chemicalElement>, <isotope>, etc.

Bibliography: adopt BibTeXML?

- BibTeXML already exists, BibTeX → BibTeXML translators already exist.
- BibTeXML includes support for DOI, urls, lots more. It appears to support all BibTeX options.

PoPs supports grouping similar particles together so common properties only need to be listed once.

- A particleGroup contains a list of similar particles. It also defines any properties that are common to all those particles.
- particle groups can be nested (see example next slide). A particleGroup must contain at least one particleGroup and/or particle.
- Do particle groups get a unique id like particles?

particleGroup examples:

- Currently the only examples of ‘particleGroup’ elements are the chemicalElement and isotope (with isotope nested inside chemicalElement):

```
<chemicalElement name=“Lithium” symbol=“Li” Z=“3”>
    <qualifiers>...</qualifiers>
    <isotope name=“Li6” A=“6”>
        <mass>...</mass>
        <nuclearLevels>
            <nuclearLevel index=“0” id=“Li6_e0”> ... </nuclearLevel>
            <nuclearLevel index=“1” id=“Li6_e1”> ... </nuclearLevel>
            ...
        </nuclearLevels>
    <isotope id=“Li7” A=“7”> ... </isotope>
</chemicalElement>
```

particleGroup examples, with aliases linking from 'Li6' to its ground state:

- Currently the only examples of 'particleGroup' elements are the chemicalElement and isotope (with isotope nested inside chemicalElement):

```
<chemicalElement name="Lithium" symbol="Li" Z="3">
   <qualifiers>...</qualifiers>
   <isotope name="Li6" A="6">
      <mass>...</mass>
      <nuclearLevels>
         <nuclearLevel index="0" id="Li6_e0" alias="Li6"> ... </nuclearLevel>
         <nuclearLevel index="1" id="Li6_e1"> ... </nuclearLevel>
         ...
      </nuclearLevels>
   <isotope id="Li7" A="7"> ... </isotope>
</chemicalElement>
```

particleGroup examples, with aliases linking from 'Li6' to its ground state:

- Currently the only examples of 'particleGroup' elements are the chemicalElement and isotope (with isotope nested inside chemicalElement):

```
<chemicalElement name="Lithium" symbol="Li" Z="3">
   <qualifiers>...</qualifiers>
   <isotope name="Li6" A="6">
      <mass>...</mass>
      <nuclearLevels>
         <nuclearLevel index="0" id="Li6_e0" alias="Li6"> ... </nuclearLevel>
         <nuclearLevel index="1" id="Li6_e1"> ... </nuclearLevel>
         ...
      </nuclearLevels>
   <isotope id="Li7" A="7"> ... </isotope>
</chemicalElement>
```

- each isotope inherits charge 'Z' from parent chemicalElement
- each nuclearLevel inherits ground-state mass from parent isotope

Should we also allow rotationalBand nested inside nuclearLevels?

particleGroup examples:

- Currently the only examples of 'particleGroup' elements are the chemicalElement and isotope (with isotope nested inside chemicalElement):

```
<chemicalElement name="Lithium" symbol="Li" Z="3">
   <qualifiers>...</qualifiers>
   <isotope name="Li6" A="6">
      <mass>...</mass>
      <nuclearLevels>
         <nuclearLevel index="0" id="Li6_e0" alias="Li6"> ... </nuclearLevel>
         <nuclearLevel index="1" id="Li6_e1"> ... </nuclearLevel>
         ...
      </nuclearLevels>
   <isotope id="Li7" A="7"> ... </isotope>
</chemicalElement>
```

'qualifiers' are defined inside a particle group and apply to all particles inside that group

Should we also allow rotationalBand nested inside nuclearLevels?

Particle 'qualifiers' add extra information about an instance of a particle, such as electron configurations for elements and isotopes

- Need qualifiers to keep library size manageable:
  - Uniquely identifying every nuclear level requires ~100K particles (assuming 100 elements * 10 isotopes/element * 100 levels/isotope)
  - If every electron configuration for every level is given explicitly, the number of particles grows by orders of magnitude!
    - Assuming 1000 possible electron configurations and 10 characters per name, requires 1 GB just to store the list of particle names!
  - qualifiers are an alternative: they add extra information to existing particles instead of requiring that more particles be defined.
    - Assumes that electron configuration (and binding energy) is the same for all isotopes and excited states of an element.

- Qualifiers currently can only be defined inside a particleGroup (i.e. 'chemicalElement' or 'isotope'). Should they be permitted anywhere else?

Basic particle properties

- Each particle in PoPs has at least the following:
  - id
  - mass
  - charge
  - spin
  - parity
  - halflife / width

- Some of these may be inherited from a parent. For example, isotopes inherit their charge from the ‘Z’ of their parent ‘chemicalElement’
- Particle families may require that additional properties be defined

Particle families determine attributes in addition to basic list that need to be supplied

- Examples of families
  - lepton (requires lepton number)
  - baryon (baryon number)
  - nuclearLevel (excitation energy and level index)
  - …
- Where should the photon live? ‘boson’? ‘gaugeBoson’?

Should PoPs define a <constituentParticles> element for describing what elementary particles are inside a conglomerate particle?

- <constituentParticles> contains a list of pids with multiplicities
- For example,

```
<baryon id="n" name="neutron">
    ...
    <constituentParticles>
        <constituent pid="up_quark" multiplicity="1"/>
        <constituent pid="down_quark" multiplicity="2"/>
    </constituentParticles></baryon>
```

- BUT, what about particles that are superpositions of their constituents (i.e. neutrinos)?

Each particle quantity (mass, spin, etc.) may have multiple assignments. PoPs should support storing all of them, along with 'recommendation'

- Multiple assignments come up for two reasons:
  - one value expressed in different ways (e.g. atomic mass vs. binding energy vs. neutron/proton separation energies). Schematically:

    -mass (along with recommendation of which value below to use):

    -atomic

    -bindingEnergy

    ...

  - multiple possible values if a definite assignment hasn't been made (e.g. spins and parities). Schematically:

    -spin (along with recommendation)

    -assignment #1

    -assignment #2

    ...

Each particle quantity (mass, spin, etc.) may have multiple assignments. PoPs should support storing all of them, along with 'recommendation'

- Multiple assignments come up for two reasons:
  - one value expressed in different ways (e.g. atomic mass vs. binding energy vs. neutron/proton separation energies). Schematically:

    -mass (along with recommendation of which value below to use):

    -atomic

    -bindingEnergy

    ...

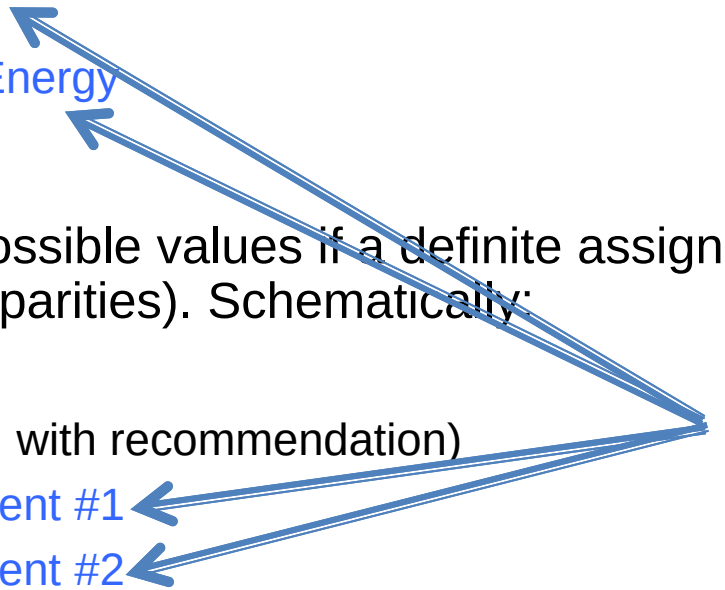
  - multiple possible values if a definite assignment hasn't been made (e.g. spins and parities). Schematically:

    -spin (along with recommendation)

    -assignment #1

    -assignment #2

    ...



Each of these single values is a 'quantity'

If a particle decays, list each decay mode separately

```
<nuclearLevel id="Mn68_e0" index="0">
    ...
    <halflife value="28" unit="ms"/>
    <decays>
      <decay label="0" mode="betaMinus">
        <probability><quantity value="..." unit=""/></probability>
        <products><product pid="Fe68"/>...</products></decay>
      <decay label="1" mode="betaDelayedNeutron">
        <probability><quantity value="..." unit=""/></probability>
        <products><product pid="n"/><product pid="Fe67"/>...</products>
      </decay></decays></nuclearLevel>
```

(nuclearLevel is a type of particle)

Some decays are related, e.g. gamma decay and internal conversion.

- Could handle these by splitting up into two separate decay modes:

  <decay label="0" mode="gamma">...</decay>

  <decay label="1" mode="internalConversion">...</decay>

- But existing libraries use internal conversion coefficient:

  <decay label="0" mode="electroMagnetic" ICC=".96"/>

- Likely need to continue supporting ICC... but outgoing products are different for IC than for gamma decay!

Decays to different excited states are treated as independent decay modes

Gamma-decay of 9th excited level in Mn55 goes to three different levels

```
<nuclearLevel id="Mn55_e9" index="9">
    <energy><quantity value="2.19843" unit="MeV"/></energy>
    <spin><quantity value="3.5" unit="hbar"/></spin><parity><quantity value="-1" unit=“”/></parity>
    <halflife value="1.8e-14" unit="s"/>
    <decays>
     <decay label="0" mode="gamma” ICC=“9.381e-5”>
      <probability><quantity value="0.33" unit=""/></probability>
      <products><product particle=“Mn55_e2"/><product particle="photon”><energy value="1.214" unit="MeV"/></product></products>
     </decay>
     <decay label="1" mode="gamma” ICC=“3.279e-5”>
      <probability><quantity value="0.06" unit=""/></probability>
      <products><product particle=“Mn55_e1"/><product particle="photon”><energy value="2.072" unit="MeV"/></product></products>
     </decay>
     <decay label="2" mode="gamma” ICC=“2.97e-5”>
      <probability><quantity value="0.61" unit=""/></probability>
      <products><product particle=“Mn55_e0"/><product particle="photon”><energy value="2.199" unit="MeV"/></product></products>
     </decay></decays></nuclearLevel>
```

Anti-particles

- Particles and their anti-particles share many properties: mass, spin and parity, halflife, etc.
- Rather than storing them as explicit particles (and duplicating these properties), allow adding an <antiParticle> element to any particle.
- <antiParticle id="..."> means adding a new particle to the database with all properties identical to the parent except for specific properties (determined by particle family)
  - for leptons, charge and lepton number are both multiplied by -1
  - for baryons, charge and baryon number multiplied by -1
  - ...
- Should rules for constructing anti-particles be explicitly stored?

Complete example of a particle: (note that id is a definition, pid is a reference)

```
<baryon id="n" name="neutron">
  <antiParticle id="n_anti"/>
  <charge><quantity value="0" unit="e"/></charge>
  <mass recommended="0">
    <quantity label="0" value="1.00866491574" unit="amu" kind="nucleon" origin="evaluated">
      <uncertainty value="5.6e-10" pdf="normal"/></quantity></mass
    <quantity label="1" value="8071.3171" unit="keV" kind="massExcess" origin="evaluated">
      <uncertainty value="5.3e-4" pdf="normal"/></quantity></mass>
  <spin><quantity label="0" value="1/2" unit="hbar"/></spin>
  <parity><quantity label="0" value="1" unit=""/></parity>
  <halflife><quantity label="0" value="611" unit="s"><uncertainty value="1" unit="s" pdf="normal"/></quantity></halflife>
  <decays>
    <decay label="0" mode="betaMinus">
      <probability><quantity label="0" value="1.0" unit="0"/></probability>
      <products><product pid="p"/><product pid="e-"/><product pid="nu_e_anti"></products>
    </decay></decays></baryon>
```

A bit more on qualifiers...

- Qualifiers are *keys* (not meant to be parsed)

- Don’t need many qualifiers to handle RIPL+AME translation (except for fully-stripped decay products like ‘d’, ‘t’ or ‘a’)
  - d = H2{’missingElectrons=1’} or H2{‘+1’}
  - a = He4{’missingElectrons=2’} or He4{‘+2’}
  - Binding energy of missing electrons is recorded in the ‘chemicalElement’ for H and He respectively

- Handling the ENDF decay and atomic relaxation sub-libraries will require qualifiers. Proposed naming:
  - Fe56{‘electronVacancy=1s1/2’;’electronVacancy=2s3/2’}
  - or should each vacancy be independent: Fe56{‘1s1/2’}{‘2s3/2’}?

Draft version of Properties of Particles database has been mostly implemented

- Python implementation of PoPs was started by postdoc Nidhi Patel. Now maintained by me
- Includes a tool for translating atomic mass evaluations (AME) and RIPL level scheme files into PoPs
- XML parser (to read PoPs files back into Python class hierarchy) is nearly complete
- Still going through some changes

Still TBD:

- Inside the reactionSuite, particle database still uses an old layout, needs to be updated to latest specifications
- Define an XML schema
- Handling ENDF decay sub-library