NuHepMC: A proposed common event format for neutrino event generators

Steven Gardiner, Joshua Isaacson, Luke Pickering

16 March 2023

Applications for a common event format

- Identified in previous workshops as valuable to the community

- Flux/geometry APIs
  - LArSoft: MCTruth (particle 4-momenta) + GTruth (extra GENIE items)
  - On the right track, but part of a big framework and missing some flexibility

- Data comparison tools
  - NUISANCE: generator inputs converted to internal FitEvent format
  - Lots of work to maintain, not used elsewhere
  - Normalizing histograms as cross sections can be tricky

- Interoperability (requires coordination beyond just a standard format)
  - Apply an FSI cascade on top of primary interactions generated elsewhere
  - Inject arbitrary new events (see talks by Marco, Leo, Alexis)

The HepMC3 standard (1)

- Widely used in other areas of high-energy physics
  - Paper: Comput. Phys. Commun. 260 (2021) 107310,
  - C++ implementation: https://gitlab.cern.ch/hepmc/HepMC3
- Several key concepts are used to represent generator output
- Run Information is common to a set of related events
  - Configuration metadata, etc.
  - Known at start of job
- Events consist of a group of Particles and Vertices
- Particles hold a 4-momentum, PDG code, and status code

The HepMC3 standard (2)

- Vertices connect sets of incoming and outgoing particles
  - Hold a 4-position and a status code
  - Encode mother/daughter relationships (must have coincident 4-positions!)

- Attributes can be attached to any of the previous objects
  - Each is a named entity with an arbitrary data type
  - Strings, integers, and floating-point numbers (and vectors of these) are officially supported

- C++ library includes classes for file input/output
  - Several formats available, including HepMC3's native text format

NuHepMC (1)

- Joint effort between the authors of this talk
- Define common standards for representing neutrino scattering events using the HepMC3 format
- Seeking to answer various questions. Some examples:
  - How can the generator configuration be stored to make a run as reproducible as possible?
  - How should interaction mode labels be handled?
  - What units should be used to represent 4-positions, 4-momenta, etc.?
  - What metadata should be included to allow events to be converted into cross sections?
- Community feedback and generator buy-in will be critical

NuHepMC (2)

- Near-final draft of a specification document
  - https://github.com/NuHepMC/Spec
  - To appear soon as a technical paper draft on arxiv
  - We will solicit this community for comments and **sign-on**
  - Then go for publication with expanded authorlist
  - Let's do this right, and make it a community standard
- Highlights will be shown on the next few slides
  - Discussion on all details is welcome
- Work on NuHepMC-compliant interfaces is also ongoing
  - Apply practical lessons learned to refining the specification
- Description of a first-pass GENIE interface later in the talk

Structure of the specification

- Define standards for 4 **components** of the HepMC3 output:
  - Generator run metadata
  - Event metadata
  - Vertices
  - Particles

- The standards are grouped into 3 **categories**:
  - Requirements (mandatory)
  - Conventions (optional but encouraged)
  - Suggestions (optional)

- Each standard is labeled by <Component>.<Category>.<Index>
  - Component ∈ { G, E, V, P }, Category ∈ { R, C, S }
  - **Example**: V.C.2 is the second convention for vertices

Application of these labels

G.C.1 Signalling Followed Conventions

To signal to a consumer that an implementation follows a named convention from this specification, a `HepMC3::VectorStringAttribute` should be added to the `HepMC3::GenRunInfo` instance named "NuHepMC.Conventions" containing the names of the conventions adhered to.

```
// Default set of implemented NuHepMC conventions
const std::set< std::string > NUHEPMC_CONVENTIONS(
  { "G.C.1", "G.C.5", "E.C.1", "E.C.6" } );

fRunInfo->add_attribute( "NuHepMC.Conventions",
  std::make_shared< HepMC3::VectorStringAttribute >( convention_vec ) );
```

Application of these labels

G.C.1 Signalling Followed Conventions

To signal to a consumer that an implementation follows a named convention from this specification, a `HepMC3::VectorStringAttribute` should be added to the `HepMC3::GenRunInfo` instance named "NuHepMC.Conventions" containing the names of the conventions adhered to.

```
// Default set of implemented NuHepMC conventions
const std::set< std::string > NUHEPMC_CONVENTIONS(
  { "G.C.1", "G.C.5", "E.C.1", "E.C.6" } );

fRunInfo->add_attribute( "NuHepMC.Conventions",
  std::make_shared< HepMC3::VectorStringAttribute >( convention_vec ) );
```

Standards for representation of interaction modes

Identifier	Process
100-199	Coherent Nuclear scattering
200-299	Quasielastic
300-399	Meson Exchange Current
400-499	Resonance production
500-599	Shallow inelastic scattering
600-699	Deep inelastic scattering
700-999	Other process types

- **E.R.2**: Each event must define an integer attribute ("ProcID") that represents the type of physics process that created it
- **G.R.4**: The run information must include
  - A list of all integers that may appear as ProcID values
  - A map that connects each integer value to a name and short description
- **E.C.1**: Defines a scheme for high-level organization of ProcID values
  - Note: no EM channels included yet

Charged current (CC) processes should have identifiers in the `X00–X49` block and neutral current (NC) in the `X50–X99` block.

Particle status codes

- **P.R.1**: Existing standards for HepMC3 should be followed (see table)
- We extend them to assign 11 == target particle (usually a nucleus)
- **G.R.6**: All generator-dependent particle status codes must be defined in the run information
- Definition storage similar to ProcID values

Status Code	Description	Usage
0	Not defined	Not meaningful
1	Undecayed physical particle	Recommended for all cases
2	Decayed physical particle	Recommended for all cases
3	Documentation line	Often used to indicate in/out particles in hard process
4	Incoming beam particle	Recommneded for all cases
5-10	Reserved for future standards	Should not be used
11	Target particle	Recommended for all cases
12-20	Reserved for future standards	Should not be used
21-200	Generator-dependent	For generator usage
201-	Simulation dependent	For simulation software usage

Cross-section information

- **E.C.5:** Cross section values should be stored in picobarns

- **E.C.2:** Event attribute ("TotXS") stores total cross section for the beam particle to interact
- **E.C.3:** Event attribute ("ProcXS") stores the total cross section for the selected ProcID

- **G.C.4:** Store the flux-averaged total cross section in the run metadata (if known at start)
  - Straightforward for simple cases (monoenergetic, flux histogram and point target)

- **E.C.4:** Store running MC estimate (and statistical uncertainty) of flux-averaged total cross section in each event
  - Likely necessary for complex fluxes and/or geometries

Draft GENIE interface (1)

- Unofficial test branch for now, but briefly discussed with other authors
  - Blame Steven G for whatever you don't like

- Adds HepMC3 library as an optional GENIE build dependency
  - **./configure --enable-hepmc3**
  - Similar to interface with external codes (INCL++, Geant4) for new FSIs in v3.2.0

```
class HepMC3Converter {

public:

  HepMC3Converter(void);

  std::shared_ptr< HepMC3::GenEvent > ConvertToHepMC3(
    const genie::EventRecord& gevrec );

  std::shared_ptr< genie::EventRecord > RetrieveGHEP(
    const HepMC3::GenEvent& evt );
```

- **genie::HepMC3Converter**
  - Bi-directional translations between genie::EventRecord objects and NuHepMC-compliant HepMC3::GenEvent objects
  - Extra GENIE event record contents stored as attributes ("GENIE.ZZZ")

Draft GENIE interface (2)

- Output in HepMC3 text-based format provided by **genie::HepMC3NtpWriter**

- Refactored gevgen command-line program
  - **gevgen -o my_ghep_events.root,ghep,my_hepmc3_events.txt,hepmc** will write equivalent output files in both formats simultaneously

- Running estimate of flux-averaged total cross section included in output (E.C.4)

- Encountered a few surprises

- **Example:** Some mother/daughter pairs do not have the same 4-position. Considering adjustments to GENIE conventions.

```
class HepMC3NtpWriter : public NtpWriterI {

public:

  HepMC3NtpWriter();
  virtual ~HepMC3NtpWriter();

  ///< initialize the ntuple writer
  virtual void Initialize() override;

  ///< add event
  virtual void AddEventRecord( int ievent, const EventRecord* ev_rec ) override;

  ///< save the event tree
  virtual void Save() override;
```

Draft GENIE interface (3)

- Test branch named **hepmc3** available on Steven G's personal GitHub fork
  - https://github.com/sjgardiner/Generator/tree/hepmc3
  - Feedback welcome
- Includes citations for active cross-section models in the run information
  - GENIE configuration XMLs edited to include Digital Object Identifiers for papers
  - This information is harvested by **genie::HepMC3Converter** at runtime
- Further work anticipated, so all implementation details are subject to change

Achilles Draft Interface

- Validated against NuHepMC Validator (https://github.com/NuHepMC/ReferenceImplementation)
- Many parts still hard coded since there is only one QE model implemented
- Still in a private branch, will be made public soon in Achilles repo

```
void NuHepMC3Writer::WriteHeader(const std::string &filename) {
    // Setup generator information
    spdlog::trace("Writing Header");
    auto run = std::make_shared<HepMC3::GenRunInfo>();
    run->add_attribute("NuHepMC.Version.Major",
                       std::make_shared<HepMC3::IntAttribute>(version[0]));
    run->add_attribute("NuHepMC.Version.Minor",
                       std::make_shared<HepMC3::IntAttribute>(version[1]));
    run->add_attribute("NuHepMC.Version.Patch",
                       std::make_shared<HepMC3::IntAttribute>(version[2]));

    struct HepMC3::GenRunInfo::ToolInfo generator={std::string("Achilles"),
                                                   std::string(ACHILLES_VERSION),
                                                   std::string("Neutrino event generator")};
    run->tools().push_back(generator);
    run->add_attribute("Achilles.RunCard",
                       std::make_shared<HepMC3::StringAttribute>(filename));
    run->set_weight_names({"CV"});

    // Add all possible processes
    // TODO Look up available processes and add bibtex info
    std::vector<int> proc_ids{101};
    run->add_attribute("NuHepMC.ProcessIDs",
                       std::make_shared<HepMC3::VectorIntAttribute>(proc_ids));
    run->add_attribute("NuHepMC.ProcessInfo[101].Name",
                       std::make_shared<HepMC3::StringAttribute>("SpectralQE"));
    run->add_attribute("NuHepMC.ProcessInfo[101].Description",
                       std::make_shared<HepMC3::StringAttribute>("Spectral function Quasielastic"));
    run->add_attribute("NuHepMC.ProcessInfo[101].Bibtex",
                       std::make_shared<HepMC3::StringAttribute>("Rocco:xxxx"));

    // List all possible vertex status codes
    // TODO Make this a conversion from enum of the EventHistory class?
    std::vector<int> vertex_ids{1};
    run->add_attribute("NuHepMC.VertexStatusIDs",
                       std::make_shared<HepMC3::VectorIntAttribute>(vertex_ids));
    run->add_attribute("NuHepMC.VertexStatusInfo[1].Name",
                       std::make_shared<HepMC3::StringAttribute>("Primary"));
    run->add_attribute("NuHepMC.VertexStatusInfo[1].Description",
                       std::make_shared<HepMC3::StringAttribute>("The main hard interaction"));
```

Conclusion

- We propose the NuHepMC standard as a common format for the neutrino generator community
  - Builds upon mature HepMC3 format used elsewhere
  - Provides guidance on representation of physics specific to neutrinos
- Work on the specification and generator interfaces continues
  - Discussion with you here and elsewhere will be very valuable going forward
  - Specification draft on GitHub: https://github.com/NuHepMC/Spec