LaTeX-ed on October 23, 2007.

HepMC 2

a C++ Event Record for Monte Carlo Generators

http://savannah.cern.ch/projects/hepmc/

User Manual Version 2.0
August 18, 2006

Lynn Garren
Fermi National Accelerator Laboratory

Abstract

HepMC 2 is an extension to the original HepMC written by Matt Dobbs. This manual is a companion to HepMC_user_manual.ps.

1 Introduction

This user manual is intended as a companion to both the online documentation[1] and the original HepMC user manual[2] written by Matt Dobbs. These manuals and the examples should provide a friendly introduction to the HepMC event record. A general overview is available in Ref. [1].

2 Why HepMC 2?

Since January 2006, HepMC has been supported as an LCG external package. The official web site is now http://savannah.cern.ch/projects/hepmc/, and compiled libraries for supported platforms are available at /afs/cern.ch/sw/lcg/external/HepMC.

Historically, HepMC has used CLHEP (Ref. [3]) Lorentz vectors. Some users wished to use a more modern Lorentz vector package. At the same time, there was concern about allowing dependencies on any external package. Therefore, the decision was made to replace the CLHEP Lorentz vectors with a minimal vector representation within HepMC.

Because this is a major change, the versioning is changed from 1.xx.yy to 2.xx.yy. Normally, a version number change in *xx* represents a change to the code and a version number change in *yy* represents a bug fix.

Changes to HepMC must be approved by the LCG simulation project.

2.1 Changes since HepMC 1.26

The HepMC_CLHEP20.h header defines several typedefs needed when compiling with CLHEP 2.0.xx.yy.

The IO_AsciiParticles class provides output in the pythia style.

GenEvent now contains pointers to a heavy ion class and a PDF information class. The pointers are set to zero by default.

The new ascii output class, IO_ExtendedAscii, will read and write the heavy ion and PDF information classes. It also reads and writes the generated mass.

Some tests have been added to the distribution (e.g., "make check" now works).

2.2 Changes as of HepMC 2.00.00

The SimpleVector.h header contains the FourVector and ThreeVector classes. GenParticle momentum and GenVertex position are represented by a FourVector. GenVertex will return the ThreeVector portion of the position. Polarization will accept or return a ThreeVector representation of the polarization.

Both FourVector and ThreeVector have templated constructors. These constructors allow you to use the GenParticle and GenVertex constructors with *any* Lorentz vector, as long as the Lorentz vector has `x()`, `y()`, `z()`, and `t()` methods.

In addition, the generated mass is now stored in GenParticle. This additional information has always been part of the HEPEVT common block. When a particle has large momentum and small mass, calculating the mass from the momentum is unreliable. Also, different machine representations and roundoff errors mean that a calculated mass is not always consistent. If no generated mass is set, then the mass is calculated from the momementum and stored in GenParticle.

[1] http://lcgapp.cern.ch/project/simu/HepMC/

[2] http://lcgapp.cern.ch/project/simu/HepMC/HepMC_user_manual.ps

Use IO_ExtendedAscii to read and write the generated mass.

3 Overview of Core Classes

3.1 HepMC::GenEvent

Important Public Methods

- **add_vertex**: *adopts the specified vertex to the event and assumes responsibility for deleting the vertex*
- **remove_vertex**: *removes the specified vertex from the event, the vertex is not deleted - thus use of this method is normally followed directly by a delete vertex operation*
- **vertex_iterator**: *iterates over all vertices in the event - described in the iterator section*
- **particle_iterator**: *iterates over all particles in the event - described in the iterator section*
- **vertex_const_iterator**: *constant version of the vertex_iterator*
- **particle_const_iterator**: *constant version of the particle_iterator*
- **print**: *gives a formatted printout of the event to the specified output stream*
- **barcode_to_particle**: *returns a pointer to the particle associated with the integer barcode argument*
- **barcode_to_vertex**: *returns a pointer to the vertex associated with the integer barcode argument*

Relevant Data Members

- **signal_process_id**: *an integer ID uniquely specifying the signal process (i.e. MSUB in Pythia).*
- **event_number**: *integer*
- **event_scale**: *(optional) the scale of this event in GeV. (-1 denotes unspecifed)*
- **alphaQCD**: *(optional) the value of the strong coupling constant α_{QCD} used for this event. (-1 denotes unspecifed)*
- **alphaQED**: *(optional) the value of the electroweak coupling constant α_{QED} (e.g. $\frac{1}{128}$) used for this event. (-1 denotes unspecifed)*
- **signal_process_vertex**: *(optional) pointer to the vertex defined as the signal process - allows fast navigation to the core of the event*
- **weights**: *a container of an arbitrary number of 8 byte floating point event weights*
- **random_states**: *a container of an arbitrary number of doubles which define the random number generator state just before the event generation*
- **heavy_ion**: *(optional) a pointer to a HeavyIon object (zero by default)*
- **pdf_info**: *(optional) a pointer to a PdfInfo object (zero by default)*

Notes and Conventions

- if hit and miss Monte Carlo integration is to be performed with a single event weight, the first weight will be used by default
- Memory allocation: vertex and particle objects will normally be created by the user with the NEW operator. Once a vertex (particle) is added to a event (vertex), it is "adopted" and becomes the responsibility of the event (vertex) to delete that vertex (particle).

The GenEvent is the container class for vertices. A listing of all vertices is maintained with the event, giving fast access to vertex information. GenParticles are accessed by means of the vertices.

Extended event features (weights, random_states, heavy_ion, pdf_info) have been implemented such that if left empty/unused performance and memory usage will be similar to that of an event without these features.

Iterators are provided as members of the GenEvent class and are described in HepMC_user_manual.ps.

The signal_process_id is packaged with each event (rather than being associated with a run class for example) to handle the possibility of many processes being generated within the same run. A container of tags specifying the meaning of the weights and random_states entries is envisioned as part of a run class - which is beyond the scope of an event record.

3.2 HepMC::GenVertex

Important Public Methods

- **add_particle_in**: *adds the specified particle to the container of incoming particles*
- **add_particle_out**: *adds the specified particle to the container of outgoing particles*
- **remove_particle**: *removes the specified particle from both/either of the incoming/outgoing particle containers, the particle is not deleted - thus use of this method is normally followed directly by a delete particle operation*
- **vertex_iterator**: *iterates over vertices in the graph, given a specified range - described in the iterator section*
- **particle_iterator**: *iterates over particles in the graph, given a specified range - described in the iterator section*

Relevant Data Members

- **position**: $\vec{x}, ct$ *stored as FourVector*
- **id**: *integer id, may be used to specify a vertex type*
- **weights**: *a container of 8 byte floating point numbers of arbitrary length, could be mapped in pairs into rows and columns to form spin density matrices of complex numbers*
- **barcode**: *an integer which uniquely identifies the GenVertex within the event. For vertices the barcodes are always negative integers.*

Notes and Conventions

- no standards are currently defined for the vertex id
- we presume that the position is in mm, but no part of HepMC enforces this
- once a particle is added, the vertex becomes its owner and is responsible for deleting the particle

The GenVertex is the container class for particles and forms the nodes which link particles into a graph structure.

3.3 HepMC::GenParticle

Important Public Methods

- **operator FourVector**: *conversion operator - resolves the particle as a 4-vector according to its momentum*
- **generatedMass**: *generated mass*
- **momentum().m()**: *calculates mass from momentum*

Data Members

- **momentum**: $\vec{p}, cE$ *stored as FourVector*
- **generated_mass**: *generated mass for this particle*
- **pdg_id**: *unique integer ID specifying the particle type*
- **status**: *integer specifying the particle's status (i.e. decayed or not)*
- **flow**: *allows for the storage of flow patterns (i.e. color flow), refer to Flow class*
- **polarization**: *stores the particle's polarization as* (θ, ϕ)*, refer to Polarization class*
- **production_vertex**: *pointer to the vertex where the particle was produced, can only be set by the vertex*
- **end_vertex**: *pointer to the vertex where the particle decays, can only be set by the vertex*
- **barcode**: *an integer which uniquely identifies the GenParticle within the event. For particles the barcodes are always positive integers.*

Notes and Conventions

- we presume that the momentum is in GeV, but no part of HepMC enforces this
- the particle ID should be specified according to the PDG standard [4]
- status codes are as defined for HEPEVT [2][3]

[3] For convenience the HEPEVT standard status codes are enumerated:

0	null entry
1	existing entry - not decayed or fragmented, represents the final state as given by the generator
2	decayed or fragmented entry
3	documentation line
4-10	undefined, reserved for future standards
11-200	at the disposal of each model builder - equivalent to a null line
201-	at the disposal of the user, in particular for event tracking in the detector

The particle is the basic unit within the event record. The GenParticle class is composed of the FourVector, Flow, and Polarization classes.

Pointers to the particle's production and end vertex are included. In order to ensure consistency between vertices/particles - these pointers can only be set from the vertex. Thus adding a particle to the particles_in container of a vertex will automatically set the end_vertex of the particle to point to that vertex.

3.3.1 HepMC::Polarization

Relevant Data Members

- **theta**: θ *angle in radians* $0 \leq \theta \leq \pi$
- **phi**: ϕ *angle in radians* $0 \leq \phi < 2\pi$

Notes and Conventions

- the angles are robust - if you supply an angle outside the range, it is properly translated (i.e. 4π becomes 0)

Polarization is a data member of GenParticle - its use is optional. It stores the (θ, ϕ) polarization information which can be returned as a ThreeVector as well.

3.4 HepMC::IO_BaseClass

Important Public Methods

- **write_event**: *writes out the specified event to the output strategy*
- **read_next_event**: *reads the next event from the input strategy into memory*
- **operator<<,operator>>**: *overloaded to give the same results as the above methods*

IO_BaseClass is the abstract base class defining the interface and syntax for input and output strategies of events and particle data tables.

Several IO strategies are supplied:

- **IO_Ascii** reads and writes events to files in machine readable ascii, thereby providing a form of persistency for the event record. This class does not process HeavyIon or PdfInfo. Events may be contained within the same file together with an unlimited number of comments. The majority of examples listed in Section 4 make use of this class.
- **IO_AsciiParticles** writes events to files in machine readable ascii using a format similar to that used by pythia.
- **IO_ExtendedAscii** reads and writes events to files in machine readable ascii, thereby providing a form of persistency for the event record. This class differs from IO_Ascii only in that it also reads and writes HeavyIon and PdfInfo. Events may be contained within the same file together with an unlimited number of comments.
- **IO_HEPEVT** reads and writes events to/from the Fortran HEPEVT common block. It relies on a helper class HEPEVT_Wrapper which is the interface to the common block (which is defined in the header file HEPEVT_Wrapper.h[4]). This IO strategy provides the means for interfacing to Fortran event generators. Other strategies which interface directly to the specific

[4]Different conventions exist for the fortran HEPEVT common block. 4 or 8-byte floating point numbers may be used, and the number of entries is often taken as 2000 or 4000. To account for all possibilities the precision (float or double) and number of entries can be set for the wrapper at run time,

i.e. HEPEVT_Wrapper::set_max_number_entries(4000);
HEPEVT_Wrapper::set_sizeof_real(8); .

To interface properly to HEPEVT and avoid nonsensical results, it is essential to get these definitions right *for your application.*

event record of a generator could be easily implemented in this style. An example of using IO_HEPEVT to transfer events from Pythia into HepMC is given in `example_MyPythia.cc`.

4 Examples

Examples are in the examples directory of the package and are installed in the installation directory under examples/HepMC.

- **Using the HepMC vertex and particle iterators:** example_UsingIterators.cc
- **Using HepMC with Pythia:** example_MyPythia.cc, example_MyPythiaOnlyToHepMC.cc, example_MyPythiaRead.cc, example_MyPythiaWithEventSelection.cc, and example_PythiaParticle.cc
- **Using HepMC with Herwig:** example_MyHerwig.cc
- **Event selection:** example_MyPythiaWithEventSelection.cc
- **Write an event file and then read it:** example_MyPythiaRead.cc
- **Build an event from scratch:** example_BuildEventFromScratch.cc

References

[1] M. Dobbs and J.B. Hansen, "The HepMC C++ Monte Carlo Event Record for High Energy Physics", Computer Physics Communications (to be published) [ATL-SOFT-2000-001].

[2] L. Garren, "StdHep 5.05 Monte Carlo Standardization at FNAL," Fermilab PM0091. Available from http://cepa.fnal.gov/psm/stdhep/.

[3] "A Class Library for High Energy Physics," (CLHEP). Available from http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/.

[4] W.-M. Yao *et al.*, "Review of particle physics," Journal of Physics **G33**, 1 (2006). Available from http://pdg.lbl.gov/.