

HepMCVisual – an interactive browser for HepMC events

Dr. Sebastian Böser

University College London, Department of Physics and Astronomy, Gower Place, London
WC1E 6BT, UK

E-mail: sboeser@hep.ucl.ac.uk

Abstract. Within the last years, HepMC has established itself as the standard event format for simulation of high-energy physics interactions and is commonly used by all LHC experiments. At the energies of the proton-proton collisions at the LHC, a full description of the generation of these events and the subsequent interactions with the detector typically involves several thousand particles and several hundred vertices. Currently, the HepMC library only provides a text-based representation of these events.

HepMCVisual is a visualization package for HepMC events, allowing to interactively browse through the event. Intuitive user guiding and the possibility of expanding/collapsing specific branches of the interaction tree allow quick navigation and visualization of the specific parts of the event of interest to the user. Thus, it may be a valuable tool not only for physics users but also for debugging MonteCarlo event generators.

Being based on the ROOT graphics libraries, HepMC visual can be used as a standalone library, as well as interactively from the ROOT console or in combination with the HepMCBrowser interface within the ATLAS software framework.

1. Introduction

At the LHC, protons will collide at an energy of 14 TeV. Describing these interactions themselves and the interactions of the particles created in these collisions with the LHC detectors in MonteCarlo simulations is a complex task. A large number of dedicated programs is available, that will handle different parts of this simulation chain, which can be roughly divided in

- the *hard scatter*, i.e. the most energetic interaction at parton level
- the *parton shower and hadronization*, i.e. the generation of (semi-)stable particles from these partons
- and the *detector simulation*, i.e. the decay or interaction of these (semi-)stable particles with the detector.

In order to compare different models or computational implementations, a common event format is required to serve as an interface inbetween the different stages. In the past years, the HepMC event record [1, 2] has established itself as a standard for this. In there, the physics events are described using vertices and particles. Each vertex describes an interaction inbetween particles, where the temporal evolution is described by differentiating and outgoing particles. The kinematics of the interactions are stored in the four-momenta of the particles, while the particle characteristics itself (such as mass, spin, etc.) are determined using a identifier – the

Table 1. Print-out of a typical simulated LHC proton-proton collision event as provided by HepMC library (abbr.)

```

-----
GenEvent: #3230 ID=1020024 SignalProcessGenVertex Barcode: 0
Entries this event: 341 vertices, 1247 particles.
Beam Particles are not defined.
Wgts(3)=1 1 1
EventScale -1 [energy]          alphaQCD=-1    alphaQED=-1
                                GenParticle Legend
                                ( Px,      Py,      Pz,      E ) Stat  DecayVtx
-----
Vertex:   -1 ID:    0 (X,cT)=+2.06e-03,+1.30e-03,+8.51e+01,+0.00e+00
I: 1      1      2212 +0.00e+00,+0.00e+00,+7.00e+06,+7.00e+06  3      -1
O:34     3        21  -3.90e+02,-1.46e+03,+1.68e+05,+1.68e+05  3      -3
--- ... ---
Vertex:   -2 ID:    0 (X,cT)=+2.06e-03,+1.30e-03,+8.51e+01,+0.00e+00
I: 1      2      2212 +0.00e+00,+0.00e+00,-7.00e+06,+7.00e+06  3      -2
O:60     4        21  -1.32e+03,-8.39e+02,-3.76e+06,+3.76e+06  3      -4
--- ... ---
Vertex:   -3 ID:    0 (X,cT)=+2.06e-03,+1.30e-03,+8.51e+01,+0.00e+00
I: 1      3        21  -3.90e+02,-1.46e+03,+1.68e+05,+1.68e+05  3      -3
O: 2      5         4  +1.12e+04,-2.71e+03,+2.62e+04,+2.86e+04  3      -5
          25        -4  -1.16e+04,+1.25e+03,+1.39e+05,+1.39e+05  2      -16
Vertex:   -4 ID:    0 (X,cT)=+2.06e-03,+1.30e-03,+8.51e+01,+0.00e+00
I: 1      4        21  -1.32e+03,-8.39e+02,-3.76e+06,+3.76e+06  3      -4
O:16     6         -4  +4.17e+02,-3.40e+03,-4.16e+05,+4.16e+05  3      -5
          27         4  -1.02e+03,-6.84e+02,-2.61e+06,+2.61e+06  2      -18
-----

```

`pdgID` – to each different type of particle. These identifiers usually follow the numbering scheme defined by the Particle Data Group [3, 4]. The associations inbetween the particles and vertices are retained by assignment of another unique identifier – the `barcode` – to each of them, where vertices will obtain negative numbers, while particles will obtain positive numbers. An event is then a collection of vertices identified by their barcodes.

For a typical collision at the LHC, these events will hold in the order of several hundreded vertices and several thousand particles. While it is straightforward for any algorithm to process this amount of information, understanding it for a physicist may be challenging due to the sheer amount of information. Furthermore, the sole visual representation of these events through the HepMC library is provided trough a text-based output as shown in table 1.

2. Design overview

Therefore, a library has been developed providing a graphically more intuitive representation of the events. Three main design goals have been followed in the course of this development:

- **Interactivity**, i.e. to allow the user to interactively obtain information about the particles and vertices and select only the part of the full event he is interested in.
- **Intuition**, i.e. to make the browsing of the event tree intuitive by following common user interface and design standards.
- **Interface**, i.e to provide an easy-to-use interface to the user of the library

Following these design guidlines, it has been chosen to use the graphics interfaces provided by the ROOT framework [5, 6], which is most widely used in many parts of the HEP community. In order to demonstrate the user interface, a concrete example using this new library to visualize simulated events provided by the software framework of the ATLAS experiment – ATHENA [7] –has been created.

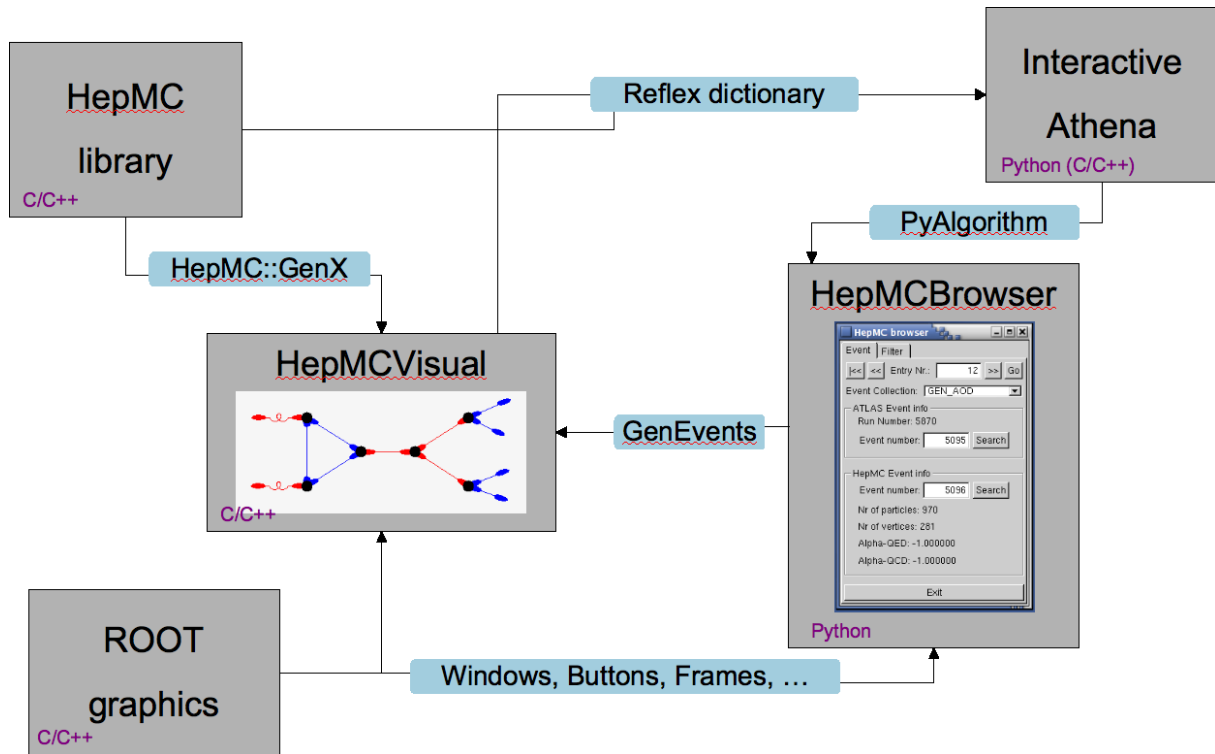


Figure 1. Design overview of the HepMCVisual library and its use in the ATHENA software framework

Figure 1 shows an overview of the design of this package and how it is used in the ATHENA software framework. The core functionality is provided by the HepMCVisual library, that links against the HepMC library to obtain the event record description and against the ROOT libraries for a graphical visualization of the objects. In order not to run in batch processing mode, but stay with a single event as long as the user wishes, ATHENA is run in the so called *interactive* mode, which provides a python command line interface. In this mode, the core functionality of the ATHENA libraries is then available via python bindings to the ROOT::Reflex [8] dictionaries of the respective classes. Hence, ROOT::Reflex dictionaries are being generated for the HepMCVisual library to make it accessible in ATHENA's interactive mode. To then completely abolish the need for any command-line interaction, an Athena algorithm is provided that establishes a GUI through which the user can browse through the events. This python algorithm – called HepMCBrowser – again makes uses of the ROOT GUI libraries via the PyROOT [9] bindings to establish a user interface.

3. Implementation

For the concrete implementation of the visual representations, some obvious and not-so-obvious choices have been made. With the HepMC events actually forming a directional graph, it is only a natural choice to use lines to represent the edges of the graph (i.e. the particles), while the nodes (i.e. the vertices) are represented by filled circles. In order to allow interactions that take into account the directionality of the graph, both ends of each edge/particle are equipped with another elliptic marker, that will be further referred to as *nibble* (c.f. figure 2, left).

One of the less obvious choices is to leave the layout of the graph (i.e. the distribution of the nodes/vertices) completely up to the user. While at first glance this may seem as an unnecessary burden, practical tests have shown that even when represented as a graph, showing all vertices and particles at the same time will not be practical due to the sheer amount of information. Developing a generic algorithm that will perform such a layout in any possible case is also highly non-trivial, and only possible at the cost of generalization. Furthermore, most users will only be interested in a very specific part of the event, such as the hard scatter, some particular decay or the hadronization stage. Thus it has been decided to leave the specific arrangement of the vertices to the user, while providing easy tools in order to navigate and constrain the displayed information to the region of interest.

3.1. Visual classes and API

With the above implementation choices in mind, a set of new classes has been written with a one-to-one correspondance to the HepMC classes they derive from.

- `VisualParticle`: `public GenParticle, public TLine`
- `VisualVertex`: `public GenVertex, public TMarker`
- `VisualEvent`: `public GenEvent`

For each of these classes, the `Draw()` method has been (over-)written to draw the object. In case of `VisualEvent`, which does not have its own graphical representation, calling `Draw()` will either draw the signal vertex (if defined in the event, e.g. using the `FindSignalVertex` method described in section 3.2) or the first vertex in the event. Deriving all these classes from their HepMC counterparts also allows to mix the visual (`VisualParticle, VisualVertex`) and logical (`GenParticle, GenVertex`) objects within the same graph. Hence the conversion from the logical to visual counterparts can happen while the user is browsing the event, thus saving memory. In order to convert a logical element to its visual counterpart, a threefold overloaded function is provided:

```
VisualParticle* HepMC::Visualize( HepMC::GenParticle* inevent);
VisualVertex* HepMC::Visualize( HepMC::GenVertex* inevent);
VisualEvent* HepMC::Visualize( HepMC::GenEvent* inevent);
```

In most cases, the later one will be the only one needed by any developer seeking to use this library. All objects can be drawn on a `TCanvas` graphics context provided by the ROOT framework, allowing to mix this information with histograms, latex formula, etc. Thus, after obtaining the `HepMC::GenEvent* pEvent` object, the actual visualization step will be as simple as:

```
new TCanvas();
Visualize(pEvent)->Draw();
```

As described below, not only the information held by the `GenParticle` itself is accessible to the user, but also the information associated with its `pdgID`. This, however, requires a particle data table to be associated with the event.

```
void VisualEvent::SetPDT(const ParticleDataTable*)
```

Since the content of this particle data table may vary considerably depending on the MonteCarlo model and from experiment to experiment, it is left to the user of the library to provide this table.

3.2. Filter functions

In order to facilitate finding the region in the graph/event the user is most interested in, a set of auxiliary functions has been developed. For ease of use, they are collected in a `VisualFilter` object, that provides the following methods:



- `bool FindSignalProcess(HepMC::GenEvent& event, const vector<int> in, const vector<int> out)`
Find a specific vertex identified by the `pdgID` of its incoming and outgoing particles. A subset of particles may be given, such as e.g only outgoing, or only one of all incoming particles. The function will return `true` on the first vertex it finds that has a matching subset of particles assigned and set this vertex as the signal vertex of the owning event. If no matching vertex is found the function returns `false`.
- `void RemovePythiaFragmentation(HepMC::GenEvent& event)`
In the Pythia event generator [10, 11] and many others, the fragmentation stage is implemented by collecting the parton shower emission in colour-neutral objects before the hadronization step. These objects are represented as `GenParticles` with `pdgIDs` in the range 91-94. This function will disconnect all particles with these `pdgIDs` from the event, as well as their parent vertices, hence leaving the event in the same stage as before the fragmentation stage.
- `void RemoveGeant(HepMC::GenEvent& event)`
In a similar way, all particles and vertices created by the widely used Geant4 detector simulation [12, 13] can be identified by their `barcode` value. Again this function will remove all vertices from the event which have been generated by Geant4.

4. Graphical User Interfaces

As outlined above, one of the main design goals is to provide intuitive user interactions with the event graph. In this section, the possibilities of the user to interact with the particles and vertices will be described, followed by a description of the HepMCBrowser GUI.

4.1. HepMCVisual user interface

One key feature needed for the intuitive understanding of physics process in an HepMC event is to be able to quickly identify particle types. In contrast to the large number of existing particles, there is only a limited set of parameters to a line that can be varied, such as the color, the thickness or the line style. A rough grouping of the particle types has been adopted, following the schematic outlined below and exemplarily summarized on the right of figure 2.

- the line-thickness indicates the number of fundamental fermions the particle is composed of, and is one for bosons.
- bosons are red, leptons are green, hadronic particles are blue, all others are black
- particles have lighter colors than their anti-particles
- neutrinos have dashed lines (- - -), gluons have curly lines (), photons have wiggly lines () and other particles have dashed-dotted lines (- . -)

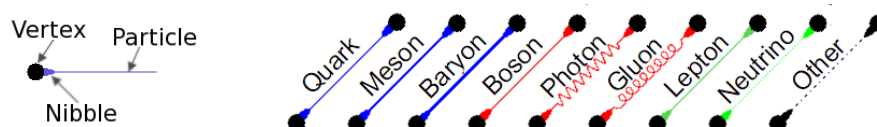


Figure 2. The visual representation (left) and examples of particle types (right)

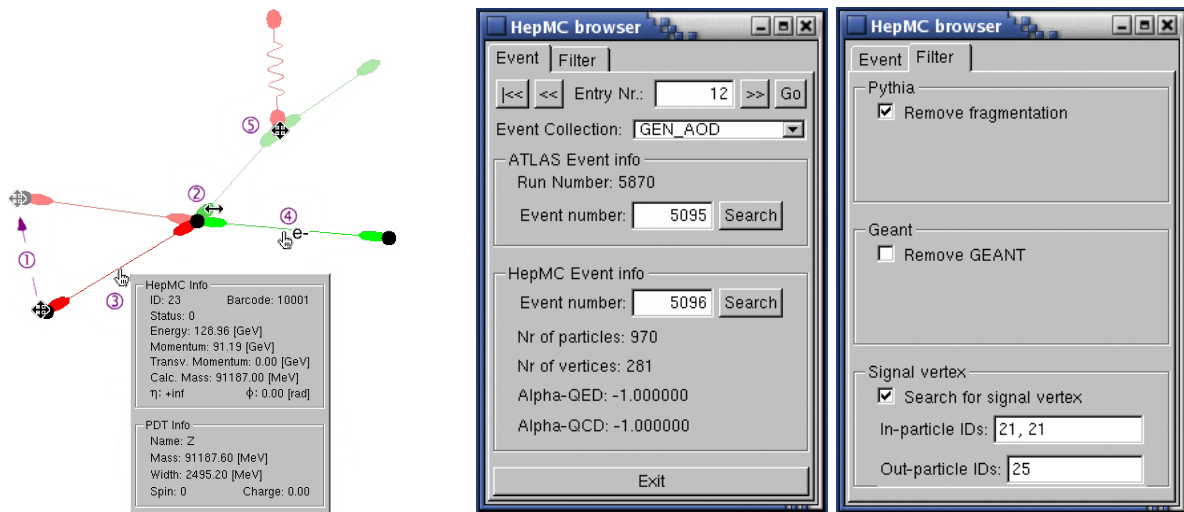


Figure 3. User interactions with the event graph (left) and HepMCBrowser GUI (right)

Once the particles and vertices are drawn, the user has five basic possibilities to interact with them as depicted on the left hand side of figure 3. When moving over the respective object, the mouse pointer will change accordingly, indicating the possible interactions.

- ① By **clicking and dragging** on a vertex, the vertex can be moved to any position on the screen. All connected particles will be automatically moved with the vertex. Vertices can not overlap each other; if the vertex is moved to a position occupied by another vertex that other vertex will be moved away. If a particle does not have an end- or production-vertex, it can still be moved **clicking and dragging** its nibbles.
- ② When **clicking on the nibble** of a particle that is attached to a vertex, that branch of the graph will be collapsed, i.e. only the nibble will remain visible. If the vertex to which the nibble is attached is an end-vertex, the upstream branch (i.e. all *parents*) will be collapsed, while for a production vertex, the downstream branch (i.e. all *children*) will be acted on. **Clicking again on the nibble** will expand the branch again. All previously collapsed / expanded branches up- or downstream of this vertex are retained throughout this operation.
- ③ While **clicking on a particle**, a popup box will appear as long as the mouse button is held down. In this popup box, all information associated with the particle will appear, as well as – if provided – all information on the particle type from the particle data table.
- ④ While **hovering** the mouse over a particle, a label with only the name of the particle type as specified in the particle data table will show up. If no particle data table has been provided, the `pdgID` will be given instead. Double-clicking on the particle will make this label persistent, allowing the user to highlight the particles of interest.
- ⑤ Finally, by **double-clicking on a vertex**, all particles associated with that vertex will be expanded. **Double-clicking** again will collapse all attached particles — not their up- or downstream branches though (otherwise the full event would disappear).

4.2. HepMCBrowser user interface

In addition, a showcase application for the HepMCVisual library has been developed within the ATHENA software framework. After providing a simulation input file, the ATHENA application can be started in interactive mode the usual way using the provided job configuration.

```
athena.py -i HepMCBrowser_jobOptions.py
```

This will create a ROOT canvas, as well as the HepMCBrowser user interface with two tabs as shown on the right side of figure 3. Depending on which of the MonteCarlo truth containers are provided in the event, the user will first have to select the appropriate input collection in the *Event* tab. Various methods of parsing through the data file – such as skipping forward or seeking for a specific event number – are provided as well. In the second tab labeled *Filter*, the user can activate one of the two filter algorithms specified in section 3.2. Alternatively, he can seek for a specific vertex within the event by specifying the pdgIDs of incoming and outgoing particles. Finally, a print button allows to print out the event in the same format as shown in table 1.

5. Availability and Documentation

Since release 15.0.0 HepMCVisual and HepMCBrowser are distributed with the ATHENA software framework. However, the HepMCVisual library is completely independent of ATHENA and can be used in any other environment. Both packages are also part of the HepForge [14] tool set and are available from

<http://projects.hepforge.org/hepmcvisual/>

The same page also features more documentation as well as a short video tutorial.

6. Acknowledgements

This work was supported by the EC Research Training Network ARTEMIS (contract number MRTN-CT-2006-035657). The author would also like to thank the maintainers of the CEDAR platform.

References

- [1] M. Dobbs and J. B. Hansen, “The HepMC C++ Monte Carlo event record for High Energy Physics,” *Comput. Phys. Commun.* **134** (2001) 41.
- [2] “HepMC - a C++ Event Record for Monte Carlo Generators,” LCG Savannah project webpage
- [3] C. Amsler et al., “Review of Particle Physics,” *Physics Letters* **B667**, 1 (2008), 333ff
- [4] “Monte Carlo Particle Numbering Scheme,” Particle Data Group webpage
- [5] R. Brun and F. Rademakers, “ROOT: An object oriented data analysis framework,” *Nucl. Instrum. Meth. A* **389** (1997) 81.
- [6] “ROOT – A data analysis framework” in ROOT webpage
- [7] P. Calafiura, W. Lavrijsen, C. Leggett, M. Marino and D. Quarrie, “The athena control framework in production, new developments and lessons learned,” *In *Interlaken 2004, Computing in high energy physics and nuclear physics* 456-458*
- [8] “Reflex – reflection for C++,” Reflex webpage
- [9] “PyROOT – python bindings for ROOT,” PyROOT webpage
- [10] T. Sjostrand, S. Mrenna and P. Skands, “A Brief Introduction to PYTHIA 8.1,” *Comput. Phys. Commun.* **178** (2008) 852 [arXiv:0710.3820 [hep-ph]].
- [11] “Pythia – a high energy physics event generator,” Pythia webpage
- [12] S. Giani *et al.*, “GEANT 4 : An Object-Oriented Toolkit for Simulation in HEP,” CERN/LHCC/98-44 (1998)
- [13] “Geant4: A toolkit for the simulation of the passage of particles through matter,” Geant4 webpage
- [14] “HepForge – a development environment for high energy physics software projects,” HepForge webpage