HepMC3 Event Record Library for Monte Carlo Event Generators
http://hepmc.web.cern.ch/hepmc/

Andrii Verbytskyi¹, Andy Buckley², David Grellscheid³, Dmitri Konstantinov⁴, Leif Lönnblad⁵, James Monk⁵,⁶, Witold Pokorski⁴ and Tomasz Przedziński⁷

ACAT 2019, Saas-Fee, Switzerland

March 13, 2019
What is HepMC3: Definition and application

- HepMC3 is a library designed to operate with Monte Carlo event records in High Energy physics (HEP).
- Event record contains all physical information on the initial, intermediate and final state particles in the simulated collision: particle flavours, momenta, production coordinates, etc.

↓

- The library should be used to store and transfer Monte Carlo event records between different HEP software and/or disk.
- HEP software in focus: Monte Carlo event generators (e.g. Pythia8[1], Herwig7 [2], SHERPA [3]), simulation software (e.g. Geant4 [4], FLUKA [5]), analysis frameworks (ROOT [6], experiment-specific), plotting tools (e.g. Rivet [7], HZTOOL [8]) etc.
The typical task is to pass MC generated events between different generators, reconstruction programs and/or analysis frameworks.

The naive expectation is that a universal translator "X library" should exist for many years, as the task is very common...

→ No universal translation library existed before 201X!

Before the HepMC3-3.0.0 the closest match was de-facto a standard HepMC2 [9] and a beta version of HepMC3.
Motivation

So why not HepMC2? Don’t fix what is not broken, HepMC2 is working so far!

Fair enough, but:

- **Maintenance:**
  - HepMC2 is not actively developed for a long time, last release is in 2012. Many major software projects and repositories should apply patches to the last version.
  - Many small changes would break compatibility but will not allow significant design improvements.

- **Features:**
  - Many physics-related issues should be updated to meet requirements of **modern physics**, e.g. Heavy Ion information [10], event weight treatment.
  - I/O capabilities should be extended to meet requirements of **modern physicists**, e.g. ROOT, LHEF [11], serialisation of custom information.

**HepMC3 is a natural successor of HepMC2. With an idea to keep is good in HepMC2 and add more.**
What was added in HepMC3 (vs. HepMC2)

- General features and I/O:
  - Consistent I/O for all popular existing formats of event records.
  - LHEF routines by Leif Lönnblad
  - ROOT interface
  - Simpler event record that allows extensions via attributes
  - Search engine to navigate in the event record

- Programming and interfaces:
  - Multithreading/thread safety
  - CMake instead of autotools should make integration easier
  - "const correctness" for the library
  - Smart pointers everywhere

- Quality assurance:
  - Continuous integration (64-bit CentOS7 and Ubuntu18).
  - Test engine based on CTest with up to 40 tests. Valgrind included.
  - Ticket system.

- Distribution
  - Closer collaboration with MCEG authors, packagers and (Linux) distributive maintainers.
Attributes in HepMC3

- An attribute is a class that holds information on event, particle or vertex and can be represented as a string.
- Every attribute class should inherit from HepMC3::Attribute. The attribute is handled all the time as a string (in memory and in I/O operations) until an explicit request to convert it to an object of some type.

Example of attributes are given below:

```
1 GenCrossSection 2.64422551e+03 2.64422551e+03 -1 -1
```

```
1 GenPdfInfo 11 -11 9.97420767e-01 9.99999975e-01 9.18812775e+01 1.56824725e+01 2.82148362e+06 0 0
```

```
alphaQCD 0.129844
```

These attributes represent information of classes:

```
1 HepMC3::GenCrossSection
1 HepMC3::GenPdfInfo
3 double
```

With attributes one can add arbitrary complex information to event record.
What was removed from HepMC3 (vs. HepMC2)

- General features and I/O
  - Some duplicated functions.
  - Operators for input/output.
  - (Custom) iterator-based access.
  - Flow and Polarisation classes are removed and the information should be represented by generic attributes. Event record is simplified!

Particle in standard HepMC2 representation (IO_GenEvent)

| 1 | P 10001 11 0.0e+00 0.0e+00 4.59999e+01 4.60000e+01 5.10999e-04 4 0 0 -1 0 |

Particle in standard HepMC3 representation (Asciiv3)

| 1 | P 1 0 11 0.0e+00 0.0e+00 4.59999e+01 4.60000e+01 5.10999e-04 4 |

The difference stands for polarization and event flow information → rarely filled and meaningful.

| 0 0 -1 0 |
I/O: ASCII formats

Listing 8: HepMC3 Asciiv3 (HepMC3 standard)

Listing 9: IO_GenEvent (HepMC2 standard) as implemented in HepMC3

Listing 10: HepEVT (Fortran era generators) as implemented in HepMC3

We keep compatibility!

Note absence of unused color flow information in Asciiv3.
I/O: LHEF

- The original Les Houches accord event format (LHEF) is designed for communicating between Matrix element generators and MCEG.
- It agreed upon in 2001 [12](see updates in Refs. [11][13][14]).
- The routines to handle LHEF is a precious part of HepMC3.

LHEF is basically XML with extra rules.

More: https://hepmc.web.cern.ch/hepmc/group__LHEF.html
I/O: ROOT Tree format

HepMC3 uses custom streamers + POD types for ROOT I/O. HepMC3 ROOT files are readable with standard ROOT, NO externals needed.
I/O: code example

```cpp
#include "HepMC3/GenEvent.h"
#include "HepMC3/WriterAscii.h"
#include "HepMC3/ReaderAsciiHepMC2.h"

int main()
{
    HepMC3::ReaderAsciiHepMC2 inputA("inputIO1.hepmc");
    if(inputA.failed()) return 1;
    HepMC3::WriterAscii outputA("frominputIO1.hepmc");
    if(outputA.failed()) return 2;
    while( !inputA.failed() )
    {
        HepMC3::GenEvent evt(HepMC3::Units::GEV,HepMC3::Units::MM);
        inputA.read_event(evt);
        outputA.write_event(evt);
        evt.clear();
    }
    inputA.close();
    outputA.close();
    return 0;
}
```

Listing 13: Reading file in HepMC2 format (IO_GenEvent) and writing in HepMC3 (Asciiv3)

- **Note:** Works in the same way for all formats in different combinations.
Example of event analysis in ROOT *(Headers omitted)*.

```cpp
class SomeAnalysis {
public :
    TChain * fChain; // pointer to the analyzed TTree or TChain
    Int_t event_number;
    Int_t momentum_unit;
    TBranch * b_hepmc3_event_event_number; // !
    TBranch * b_hepmc3_event_momentum_unit; // !
    TBranch * b_hepmc3_event_length_unit; // !
    TBranch * b_hepmc3_event_particles_; // !

    void Init(TChain *tree) {
        if (!tree) return;
        fChain = tree;
        fChain->SetMakeClass(1);
        fChain->SetBranchAddressAddress("event_number", &event_number, &b_hepmc3_event_event_number);
    }

    SomeAnalysis(const std::string & file) {
        TChain* TempChain = new TChain("hepmc3_tree");
        TempChain->Add(file.c_str());
        Init(TempChain);
    }
};

int main() {
    TH1D H1("H1","Pt of pions or electrons;Events/100MeV;P_{T},GeV",1000,0,100);
    SomeAnalysis* A = new SomeAnalysis("inputIO4.root");
    if (!A->fChain->GetEntries()) return 1;
    for (int entry=0; entry<A->fChain->GetEntries(); entry++){
        A->fChain->GetEntry(entry);
        for (int i=0; i<A->particles_; i++)
            if (A->particles_status[i]==1&&((A->particles_pid[i])==211||A->particles_pid[i]==11))
                H1.Fill(std::sqrt(A->particles_momentum_m_v1[i]*A->particles_momentum_m_v1[i] + A->particles_momentum_m_v2[i]*A->particles_momentum_m_v2[i]));
    }
    delete A;
    H1.Print("All");
    return 0;
}
```

- Reading file in ROOT format without HepMC3.
- Version with HepMC3 is in backups.
Custom classes

HepMC3 I/O is easily extendable for different purposes

- See **30 LOCs** to write events for ZEUS detector simulation.
- Writing events into dot format is given in git repository. To be used with GraphViz:

**Figure:** $e^- p$ collision in Herwig 7.1.4

**Figure:** $e^- p$ collision in Pythia 8.2.40
Format conversion

convert_example 3.1
Convert between different file formats of Monte Carlo event record.
Example:
    convert_example -i hepmc2 -o treeroot input.hepmc output.root
Usage: convert_example [OPTIONS]... [inputfile outputfile]...

-h, --help
    Print help and exit

-i, --input-format=STRING
    Input format (possible values="hepmc2", "hepmc3", "hpe", "root", "treeroot", "lhef") (mandatory)

-o, --output-format=STRING
    Output format (possible values="hepmc2", "hepmc3", "hpe", "root", "treeroot", "treerootopal", "hpezeus", "dump", "dot") (mandatory)

-e, --extensions=STRING
    Extensions, in a form extension=value, could be passed to readers/writers

--events-limit=LONG
    Limit of events to read from input
    (default='100000000')

--first-event-number=LONG
    Lowest allowed event number
    (default='100000000')

--last-event-number=LONG
    Highest allowed event number
    (default='100000000')

--print-every-events-parsed=LONG
    Frequency of parsing information printouts
    (default='100')

convert_example.exe is provided in examples.
Can be recompiler with custom classes.
Note: "hpezeus", "dot", "treerootopal" are custom classes.
The main class to perform selection is HepMC3::Selector.

```cpp
def ConstSelectorPtr status = std::make_shared<SelectorWrapper<int>>([](ConstParticlePtr p)->int{return p->status();});
def ConstSelectorPtr pt = std::make_shared<SelectorWrapper<double>>([](ConstParticlePtr p)->double{return p->momentum().pt();});
```

One can then use the Selector to construct Filter functions that evaluate on particles, e.g.

```cpp
def Filter is_stable = (*status) == 1;
def bool stable = is_stable(p);
def bool beam = (*status == 4)(p);
```

Selector contains a few standard Selectors already defined, e.g.

```cpp
def ConstGenParticlePtr p;
(Selector::STATUS == 1)(p);
def (Selector::PT > 15.)(p);
(abs(Selector::RAPIDITY) < 2.5)(p);
```

One can also combined them e.g.

```cpp
def Filter myCuts = (Selector::PT > 15.) && (*abs(Selector::RAPIDITY) < 2.5) || (Selector::PT > 100.);
def bool passCuts = myCuts(p);
```

Advantages of C++11 in action
GenHeavyIon is a standard attribute to hold information in Heavy Ion collisions.

Significantly expanded in comparison to HepMC2.

Can be extended even more, e.g. to comply to Lisbon Accord [10] or needs of Heavy Ion experiments.

```cpp
int Ncoll_hard; /// the number of hard nucleon-nucleon collisions.
int Npart_proj; /// the number of participating nucleons in the projectile.
int Npart_targ; /// the number of participating nucleons in the target.
int Ncoll; /// the number of inelastic nucleon-nucleon collisions.
int N_Nwounded_collisions; /// Collisions with a diffractively excited target nucleon.
int Nwounded_N_collisions; /// Collisions with a diffractively excited projectile nucleon.
int Nwounded_Nwounded_collisions; /// Non-diffractive or doubly diffractive collisions.
double impact_parameter; /// The impact parameter.
double event_plane_angle; /// The event plane angle.
double sigma_inel_NN; /// The assumed inelastic nucleon-nucleon cross section
double centrality; /// The centrality.
double user_cent_estimate; /// A user defined centrality estimator.
int Nspec_proj_n; /// The number of spectator neutrons in the projectile
int Nspec_targ_n; /// The number of spectator neutrons in the target
int Nspec_proj_p; /// The number of spectator protons in the projectile
int Nspec_targ_p; /// The number of spectator protons in the target
map<int,double> participant_plane_angles; /// Participant plane angles
map<int,double> eccentricities; /// Eccentricities
```
## Current usage

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHERPA-MC</td>
<td>MCEG</td>
<td>3.1/3.0 in SHERPA-MC master/2.2.5</td>
</tr>
<tr>
<td>JetScape</td>
<td>MCEG</td>
<td>3.0 in JetScape 1.0</td>
</tr>
<tr>
<td>ThePEG</td>
<td>MCEG Toolkit</td>
<td>3.1 in ThePEG master</td>
</tr>
<tr>
<td>Herwig7</td>
<td>MCEG</td>
<td>3.1 via ThePEG</td>
</tr>
<tr>
<td>Pythia8</td>
<td>MCEG</td>
<td>3.1/3.0 in HepMC3</td>
</tr>
<tr>
<td>Pythia6</td>
<td>MCEG</td>
<td>3.1 in HepMC3 examples</td>
</tr>
<tr>
<td>Tauola</td>
<td>MCEG</td>
<td>3.1/3.0 in HepMC3</td>
</tr>
<tr>
<td>Photos</td>
<td>MCEG</td>
<td>3.1/3.0 in HepMC3</td>
</tr>
<tr>
<td>WHIZARD</td>
<td>MCEG</td>
<td>3.0?</td>
</tr>
<tr>
<td>EvtGen</td>
<td>MCEG</td>
<td>in touch with authors</td>
</tr>
<tr>
<td>GeantV</td>
<td>Simulation</td>
<td>3.0</td>
</tr>
<tr>
<td>pyhepmc-ng/scikit-hep</td>
<td>Utility</td>
<td>3.1/3.0 in scikit-hep master</td>
</tr>
<tr>
<td>MC-ANALYSER</td>
<td>Analysis/Ploting</td>
<td>HepMC3</td>
</tr>
<tr>
<td>Rivet</td>
<td>Analysis/Ploting</td>
<td>work in progress</td>
</tr>
</tbody>
</table>

Note: HepMC3 3.1.0 and HepMC2 can co-exist in one installation → painless migration from HePMC2. We will help you to find out how to implement HepMC3 support → hepmc-dev@cern.ch.
Requirements

Minimal:
- Modern Linux, OSX or Windows system
- C++11 compatible compiler
- CMake 2.8.X

Recommended=Minimal+
- CMake> 3.3.4
- ROOT6

Full=Recommended+
- doxygen, latex, graphviz for documentation
- Pythia8, HepMC2, Photos [15], Tauola [16], MC-TESTER [17] for tests
- Fortran77 compiler for examples

Contact developers in case something is not working.
Hopefully, there will be no need to know the requirements.
# Adoption by major distributives

<table>
<thead>
<tr>
<th>OS</th>
<th>Tested</th>
<th>Repository</th>
<th>HepMC3 versions</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacOS</td>
<td>+</td>
<td>homebrew-hep</td>
<td>3.1</td>
<td>Enrico Bothmann</td>
</tr>
<tr>
<td>ArchLinux</td>
<td>-</td>
<td>Standard (AUR)</td>
<td>3.1/3.0</td>
<td>Frank Siegert</td>
</tr>
<tr>
<td>Debian9 (+Ubuntu18, ...)</td>
<td>+</td>
<td>Standard</td>
<td>(3.1 In progress)</td>
<td>Mo Zhou</td>
</tr>
<tr>
<td>Fedora28,29,30,31 (+CentOS7,RHEL7, ...)</td>
<td>+</td>
<td>Standard</td>
<td>3.1</td>
<td>Mattias Ellert</td>
</tr>
<tr>
<td>openSUSE42.3 (+SUSE, ...)</td>
<td>+</td>
<td>Multiple unofficial</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>Windows10</td>
<td>+</td>
<td>N/A</td>
<td>3.1</td>
<td></td>
</tr>
</tbody>
</table>

One should be able to do

```bash
[username@fedora ~]$ dnf install HepMC3
or
[username@centos ~]$ yum install HepMC3
or
username—macbook% brew install HepMC3
```

without knowing the requirements. **Ignorance is Strength.**

We want to save your time and ensure painless installation.
Summary

- HepMC3 version 3.1 was released.
- Moving towards full replacement of HepMC2.
- Collection suggestions from LHC experiments, MCEG authors and interested individuals.

Visit http://hepmc.web.cern.ch/hepmc/

- Thanks a lot for the feedback we got recently! The fixes will enter the version 3.1.1 soon.
- Special thanks to people who packaged HepMC3 for distributives: Enrico Bothmann, Mattias Ellert, Frank Siegert and Mo Zhou.
Further developments

- Under discussion/implementation:
  - Python bindings
  - Module with extra attributes, e.g. to hold information on TMDs, Heavy Ion observables, DVCS observables, polarisation of beams, etc.

- Random ideas:
  - Visualisation module, e.g. like VisualHepMC. Largely covered by WriterDOT from examples.
  - PDG information module and utils, e.g. like in HepMC<2.05. Either trivial (masses and widths), does not exist (branching fractions) or depends on used software (particle codes).

- Your suggestions are very welcome!
# include "HepMC3/GenEvent.h"
# include "HepMC3/GenParticle.h"
# include "HepMC3/ReaderRootTree.h"
#include <TH1D.h>

int main()
{
    TH1D H2("H2","Pt of pions or electrons;Events/100MeV;P_{T},GeV",1000,0,100);
    HepMC3::ReaderRootTree inputA("input104.root");
    if(inputA.failed()) return 10002;
    while(!inputA.failed())
    {
        HepMC3::GenEvent evt(HepMC3::Units::GEV,HepMC3::Units::MM);
        inputA.read_event(evt);
        if(inputA.failed()) {printf("End of file reached. Exit.\n"); break;}
        for(auto p: evt.particles())
        {
            if( std::abs(p->status()) == 1 && (std::abs(p->pdg_id()) == 211||std::abs(p->pdg_id()) == 11) )
                H2.Fill(p->momentum().perp());
            evt.clear();
        }
    }
    inputA.close();
    H2.Print("All");
    return 0;
}

Listing 22: Reading file in ROOT format with HepMC3
MC events fro ZEUS simulation

One of options for MC simulations in ZEUS was to use ASCII input very similar to HEPEVT format. The implementation of format in HepMC3 is listed below.

```cpp
#include " HepMC3 / WriterHEPEVT .h"
namespace HepMC3 {
  class WriterHEPEVTZEUS : public WriterHEPEVT {
    public:
      WriterHEPEVTZEUS( const std::string & filename);
      void write_hepevt_event_header();
      void write_hepevt_particle( int index, bool iflong );
  };
}
Listing 23: Header
```

```cpp
#include "WriterHEPEVTZEUS.h"
#include "HepMC3/HEPEVT_Wrapper.h"
namespace HepMC3{
  WriterHEPEVTZEUS::WriterHEPEVTZEUS( const std::string & filename): WriterHEPEVT(filename) {}
  void WriterHEPEVTZEUS::write_hepevt_event_header() {
    fprintf(m_file," E %12i %12i %12i\n",HEPEVT_Wrapper::event_number(),0,HEPEVT_Wrapper::number_entries());
  }
  void WriterHEPEVTZEUS::write_hepevt_particle( int index, bool /* iflong */ ){
    fprintf(m_file," %12i %8i\n", HEPEVT_Wrapper::status(index), HEPEVT_Wrapper::id(index));
    fprintf(m_file," %8i %8i\n", HEPEVT_Wrapper::first_parent(index), HEPEVT_Wrapper::last_parent(index));
    fprintf(m_file," %8i %8i\n", HEPEVT_Wrapper::first_child(index), HEPEVT_Wrapper::last_child(index));
    fprintf(m_file," % 19.11 E% 19.11 E% 19.11 E% 19.11 E% 19.11 E\n", HEPEVT_Wrapper::px(index), HEPEVT_Wrapper::py(index), HEPEVT_Wrapper::pz(index), HEPEVT_Wrapper::e(index), HEPEVT_Wrapper::m(index));
    fprintf(m_file, "% -52s% 19.11 E% 19.11 E% 19.11 E% 19.11 E\n", HEPEVT_Wrapper::x(index), HEPEVT_Wrapper::y(index), HEPEVT_Wrapper::z(index), HEPEVT_Wrapper::t(index),0.0);
  }
}// namespace HepMC3
Listing 24: Source
```
Installation on CentOS7

```bash
[root@hostname1 username]# yum install HepMC3 HepMC3-devel HepMC3-search HepMC3-search-devel HepMC3-rootIO HepMC3-rootIO-devel HepMC3-interfaces-devel HepMC3-doc
```

2 Resolving Dependencies
Skipping filters plugin, no data
4 → Running transaction check
--- Package HepMC3-x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-devel-x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-doc-search x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-interfaces-devel x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-rootIO x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-rootIO-devel x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-search x86_64 0:3.1.0-3.x7 will be installed
--- Package HepMC3-search-devel x86_64 0:3.1.0-3.x7 will be installed
→ Finished Dependency Resolution
16 Dependencies Resolved
18 Package Arch Version Repository Size
20 Installing:
```
| HepMC3    | x86_64 | 3.1.0-3.x7 | epel-testing | 258 k |
| HepMC3-devel | x86_64 | 3.1.0-3.x7 | epel-testing | 52 k |
| HepMC3-doc | noarch | 3.1.0-3.x7 | epel-testing | 43 k |
| HepMC3-rootIO | x86_64 | 3.1.0-3.x7 | epel-testing | 63 k |
| HepMC3-rootIO-devel | x86_64 | 3.1.0-3.x7 | epel-testing | 5.5 k |
| HepMC3-search | x86_64 | 3.1.0-3.x7 | epel-testing | 29 k |
| HepMC3-search-devel | x86_64 | 3.1.0-3.x7 | epel-testing | 10 k |
```
24 Transaction Summary
```
Transaction completed successfully.
```
26 Complete

New leaves:
- HepMC3-doc
- HepMC3-rootIO-devel
- HepMC3-search-devel
- HepMC3-devel
Bibliography I


Bibliography II

[8] J. Bromley et al.,
HZTOOL: A package for Monte Carlo-data comparison at HERA (version 1.0),

[9] M. Dobbs and J.B. Hansen,
The HepMC C++ Monte Carlo event record for High Energy Physics.

[10] J.G. Milhano et al.,
Lisbon Accord: a standard format for heavy-ion event generators.
(2017).

A Standard format for Les Houches event files.

[12] E. Boos et al.,
Generic user process interface for event generators.

(2010).
Bibliography III

[14] J.R. Andersen et al.,
(2014).

[15] E. Barberio and Z. Was,

[16] S. Jadach, J.H. Kühn and Z. Was,
TAUOLA: a library of Monte Carlo programs to simulate decays of polarized tau leptons.

[17] P. Golonka, T. Pierzchala and Z. Was,
MC-TESTER: A Universal tool for comparisons of Monte Carlo predictions for particle decays in high-energy physics.