Table of Contents

Data Formats in GeneratorInterface

Event content definition

RECO Data Formats

InputTag/Module (Instance name)	Containers	Description
GeneratorInterface collections (in RECOSIM and AODSIM)		
generator	LHEEventProduct	General characteristics of a generated event (only present if the event starts from LHE events)
generator	GenEventInfoProduct	General characteristics of a generated event.
generator	GenRunInfoProduct	Run-specific parameters that define event generation, such as cross-sections, etc.
GeneratorInterface collections (in RECOSIM only)		
generator	edm::HepMCProduct	A tree of final-state particles that form a generated event.
generator	LHEEventProduct	General characteristics of a generated event (only present if the event starts from LHE events)
generator	GenEventInfoProduct	General characteristics of a generated event.
generator	GenRunInfoProduct	Run-specific parameters that define event generation, such as cross-sections, etc.

How to use the table

In the header of your analyzer you should have the header(s) of the data format(s) you will access. Generator data formats belong to the in the SimDataFormats/GeneratorProducts; the headers are located in the `/interface` subdirectory. Thus, your **EDAnalyzer** should contain all or one of the following:

```
#include "SimDataFormats/GeneratorProducts/interface/HepMCProduct.h"
#include "SimDataFormats/GeneratorProducts/interface/GenEventInfoProduct.h"
#include "SimDataFormats/GeneratorProducts/interface/GenRunInfoProduct.h"
```

Using the directly (only works with GEN/GENSIM)

Note that `edm::HepMCProduct` is a wrapper on top of `HepMC::GenEvent` class of the standard LCG package HepMC.
For details about features and access methods of the `HepMC::GenEvent` class please refer to the HepMC documentation for details.

Here we would like to show how to access the generated event particle tree and to use 2 of the `GenEvent::HepMC` access methods.
In the `analyze(const Event& e, const EventSetup& )` method of your **EDAnalyzer** you can do something like the following:

```
edm::Handle<edm:: HepMCProduct > genEvtHandle;
e.getByLabel( "generator", genEvtHandle) ;
```

```
const HepMC::GenEvent* Evt = genEvtHandle->GetEvent() ;
//
// this is an example loop over the hierarchy of vertices
//
for ( HepMC::GenEvent::vertex_const_iterator
          itVtx=Evt->vertices_begin(); itVtx!=Evt->vertices_end(); ++itVtx )
{
      //
      // this is an example loop over particles coming out of each vertex in the loop
      //
      for ( HepMC::GenVertex::particles_out_const_iterator
              itPartOut=(*itVtx)->particles_out_const_begin();
              itPartOut!=(*itVtx)->particles_out_const_end(); ++itPartOut )
      {
         // and more of your code...
      }
}
```

Out of the event size consideration, in the AOD event content the `edm::HepMCProduct` format of the is generated event is replaced by the "lighter" record called `reco::GenParticleCollection`. Formally, it is outside the GeneratorInterface domain, and belongs to the CMS.PhysicsTools of CMSSW. For more details, please visit WorkBookGenParticleCandidate and related materials.

Using other products (AODSIM)

If you wish to analyze general characteristics of an event rather than specific generated particles, in the `analyze(const Event& e, const EventSetup& )` of your **EDAnalyzer** do the following:

```
edm::Handle<GenEventInfoProduct> genEvtInfo;
e.getByLabel( "generator", genEvtInfo );
double qScale = genEvtInfo->qScale();  // in case of Pythia6, this will be pypars/pari(23)
const std::vector<double>& binningValues = genEvtInfo->binningValues(); // in case of Pythia6, th
```

Particularly important is the use of weights, which must **always** be activated otherwise events with generators like Madgraph5_aMCatNLO will give wrong results.

```
std::vector<double>& evtWeights = genEvtInfo->weights();
double theWeight = genEvtInfo->weight();
// and some more of your code again...
```

Please note that the `weights()` return the container of the associated event weights, where there may be 1 or more elements. For example, in the case of running Pythia6 generators there'll be 2 elements; please see SWGuidePythia6Interface#Example_8_CSA_mode_with_Event_Re for more details.
The `weight()` method returns the first element of the weights container and is normally the one to be used as nominal weight (see definition of GenEventInfoProduct).

If you wish to access run-specific information about event generation, you can do so via `GenRunInfoProduct`. Please be advised that this product

- belongs to `edm::Run` rather than `edm::Event`
- is not finalized until the end of run

Thus you may want to implement `endRun(const  Run& r, const EventSetup& )` method in your **EDAnalyzer** and analyze GenRunInfoProduct there:

```
edm::Handle<GenRunInfoProduct> genRunInfo;
r.getByLabel( "generator", genRunInfo );
```

Retrieving information on LHE event (AODSIM)

Only in case the generation started from LHE events the GenEventInfo will contain information on matching like the number of original partons before hadronization:

```
int nAllPartons = genEvtInfo->nMEPartons();
int nPartonsEnteringMatching = genEvtInfo->nMEPartonsFiltered();
// and some more of your code again...
```

Also, if the LHE was produced by Madgraph5_aMCatNLO or POWHEG in official production weights from scale variations, PDFs etc. are stored in the relative product. Notice that to be used they need to be renormalized to the central event weight **at LHE level** which may be different from `genEvtInfo->weight()`.

```
edm::Handle<LHEEventProduct> EvtHandle ;
e.getByLabel( theSrc , EvtHandle ) ;

int whichWeight = XXX;
theWeight *= EvtHandle->weights()[whichWeight].wgt/EvtHandle->originalXWGTUP();
```

To know which integer XXX corresponds to which weight you can use:

```
edm::Handle<LHERunInfoProduct> run;
typedef std::vector<LHERunInfoProduct::Header>::const_iterator headers_const_iterator;

iRun.getByLabel( "externalLHEProducer", run );
LHERunInfoProduct myLHERunInfoProduct = *(run.product());

for (headers_const_iterator iter=myLHERunInfoProduct.headers_begin(); iter!=myLHERunInfoProduct.h
  std::cout << iter->tag() << std::endl;
  std::vector<std::string> lines = iter->lines();
  for (unsigned int iLine = 0; iLine<lines.size(); iLine++) {
   std::cout << lines.at(iLine);
  }
}
```

Some commonly used weights for samples:

- QCD scales variations: all combinations of mu_R x2 - x0.5 and mu_F x2 - x0.5, except mu_R x2, mu_F x0.5 and mu_R x0.5, mu_F x2 (see CitationsForGenerators)
- Members of NNPDF3.0LO or NLO (default PDF, central + 100 variations, alpha_s = 0.130). Numbers are different if the sample uses the 4-flavor version of PDFs.
- NNPDF3.0 with alpha_s varied to 0.118
- NNPDF3.0 with alpha_s varied to 0.119
- CTEQ6L1 or CT10 (alternative PDF)
- Members of MMHT14lo68cl or nlo or nnlo (alternative PDF, central + 50 variations)
- Members of HERAPDF15 (alternative PDF, central + 20 variations)

Review status

Reviewer/Editor and Date (copy from screen)	Comments
RobertoCovarelli - 2015-03-27	update
JuliaYarba - 13 Aug 2009	filled up page with basic information
KatiLassilaPerini - 22 Jul 2008	created template page

Responsible:
Last reviewed by:

This topic: CMSPublic > SWGuideDataFormatGeneratorInterface
Topic revision: r16 - 2020-04-17 - MaximilianMariaHorzela