





**LUND**  
UNIVERSITY



# THEPEG

## Toolkit for High Energy Physics Event Generation

- Introduction
- Overview
- Status & Future Plans

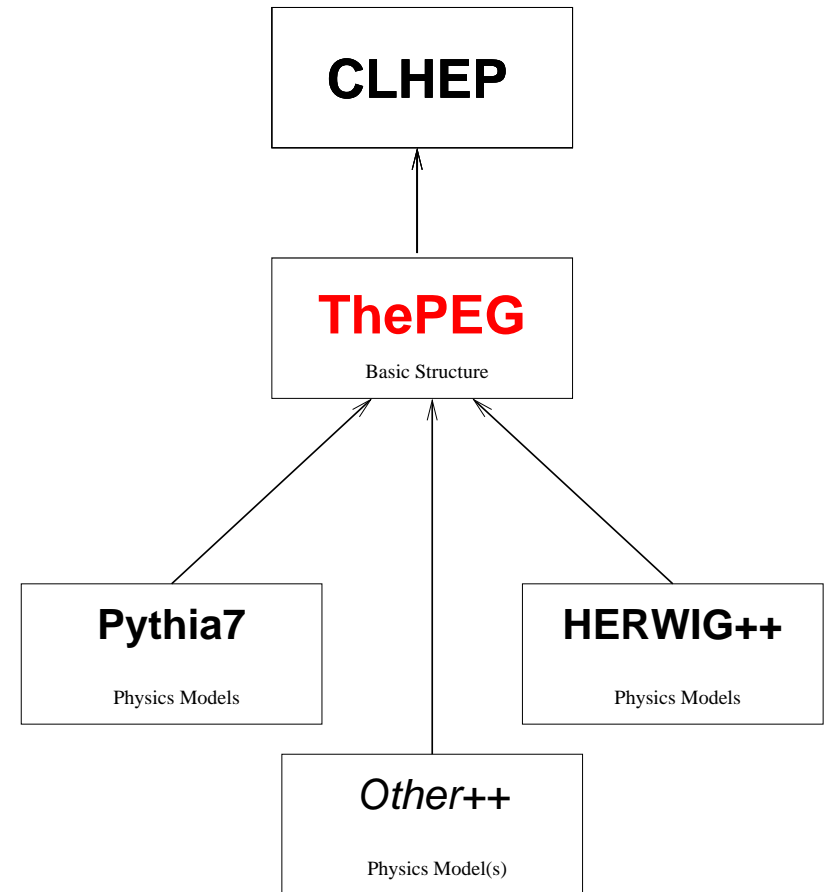
**CERN**  
**2003.07.16**  
**Leif Lönnblad**

# What is THEPEG

THEPEG consists of the parts of PYTHIA7 which were not specific to the PYTHIA physics models. It provides a general structure for implementing models for event generation.

Both PYTHIA7 and HERWIG++ are built on THEPEG.

But it is open for anyone. . .



# The components of THEPEG

- **Basic infrastructure:** Smart pointers, extended type information, object persistency, Exceptions, Dynamic loading, ...
- **Kinematics:** Extra utilities on top of CLHEP vectors, 5-vectors, flat n-body decay, ... should be moved to CLHEP.
- **Repository:** Manipulation of **interfaced** objects. Setting of parameters and switches and connecting objects together.
- **Handler classes:** to inherit from to implement a specific physics model.
- **Event record:** Used to communicate between handler classes.
- **Particle data:** particle properties, decay tables, decayers etc...



THEPEG defines a set of abstract **Handler** classes for hard partonic sub-processes, parton densities, QCD cascades, hadronization, etc. . .

These handler classes interacts with the underlying structure using a special **Event Record** and a pre-defined set of **virtual** function definitions.

The procedure to implement e.g. a new hadronization model, is to write a new (C++) class **inheriting** from the abstract **HadronizationHandler** base class, implementing the relevant virtual functions.



The structure of the generation process is extremely dynamic:

Besides the standard Handler classes, there is also a general **StepHandler** class which can do anything and can be inserted anywhere in the generation chain.

In addition, each handler can add steps in the generation chain or redo previous steps depending on the history of each event.



# How to use THEPEG

Running THEPEG is separated into two phases.

- **Setup:**

A setup program is provided to combine different objects implementing physics models together to build up an EventGenerator object. Here the user can also change parameters and switches etc. <sup>a</sup>

No C++ knowledge is needed for this. In the future we would like a nice GUI so that the user can just click-and-drag.

The [Repository](#) already contains a number of ready-built EventGenerators. It is also possible to specify AnalysisHandler object for an EventGenerator.

In the end the built EventGenerator is saved to a file.

---

<sup>a</sup>See tutorial next week.



- Running:

The saved EventGenerator can be simply read in and run using a special slave program. If AnalysisHandlers have been specified, this is all you have to do.

Alternatively the the file with the EventGenerator can be read into any program which can then use it to generate events <sup>a</sup> which can be sent to analysis or to detector simulation.

---

<sup>a</sup>ThePEG::Events which can be translated into HepMC::GenEvents



The EventGenerator class is the main class administrating an event generation run.

It maintains global information needed by the different models: The ParticleData objects to be used, a StandardModel object with couplings etc, a RandomGenerator, a list of AnalysisHandlers etc.

It also has an EventHandler object to administer the actual generation.



The **EventHandler** is a CollisionHandler which keeps a list of SubProcessHandlers <sup>a</sup> each of which associates a PartonExtractor object with a list of MEBase <sup>b</sup> objects. It also has a SampleBase object to do the phase space sampling and integration and a KinematicalCuts object to specify cuts.

The **PartonExtractor** uses PDFBase and RemnantHandler objects to generate the incoming partons. And each pair of incoming partons are combined with each MEBase object into an XComb object which is responsible for the actual generation of the hard sub-process.

---

<sup>a</sup>With two SubProcessHandlers you can generate both diffractive and non-diffractive events in the same run.

<sup>b</sup>Matrix element objects, see tutorial next week



The `CollisionHandler` is a `PartialCollisionHandler` which after a sub-process has been generated administers the application of different `StepHandler` objects divided up in groups.

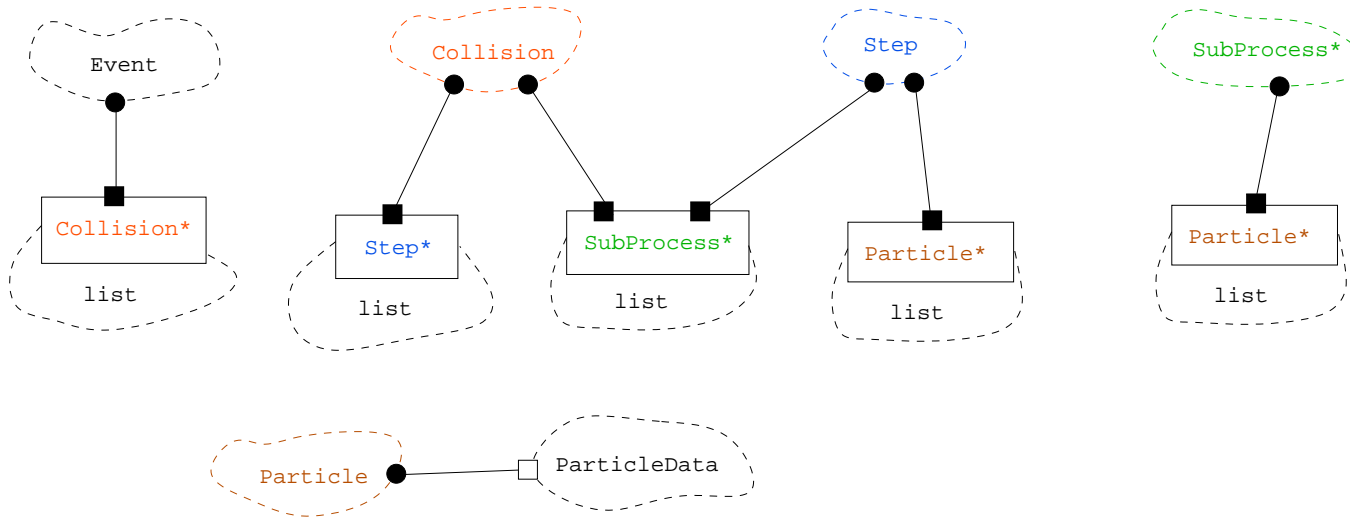
Each of the groups has a specific main handler object and a list of general `StepHandlers` to be applied before and after the main one.

The special handler classes are `CascadeHandler`, `MultipleInteractionHandler`, `HadronizationHandler` and `DecayHandler`.

The `PartialCollisionHandler` collision handler can be used separately in a `PartialEventGenerator` object if the hard sub-process has been generated from the outside.



# Class structure of an Event



A fairly complicated structure to allow for complicated analysis. But it should still be simple to do simple things:

```
PersistentIStream is("AFileWithAnEventGenerator.run");
EGPtr eg; // Open a file
is >> eg; // Read in an EventGenerator
Event ev = eg->shoot(); // generate an event...
vector<ParticlePtr> particles;
ev.getFinalState(particles); // Get the final-state particles
```



The `Particle` class provides access to a lot of information. But the class only has a pointer to a `ParticleData`, a `Lorentz5Momentum` and a pointer to another object carrying the rest of the information (colour, spin etc.) if needed.

Some of this information can be user-defined by creating classes inheriting from e.g. the `SpinBase` or the completely general `EventInfoBase` classes. This information can then be accessed through `dynamic_casting`.



# How to implement PDF's

A parton density is not just a function  $x f_j^p(x, Q^2)$ .

To add a PDF parameterization to PYTHIA7 we create a new class inheriting from the **PDFBase** class. The following abstract virtual methods must be implemented:

```
virtual bool canHandleParticle(tcPDPtr particle) const;
```

can this PDF handle the given particle?

```
virtual cPDVector partons(tcPDPtr particle) const;
```

which partons can be extracted from the given particle?



```
virtual double xfl(tcPDPtr particle, tcPDPtr parton,  
                  Energy2 partonScale, double l,  
                  Energy2 particleScale = 0.0*GeV2) const;
```

The main function giving the parton density for parton in particle at some partonScale and momentum fraction  $l = \log(1/x)$ . Also the off-shellness of the particle may be given (e.g. for virtual photon densities).

- `PDPtr` (smart) pointer to a `ParticleData` object
- `PDVector` vector of pointers to `ParticleData` objects
- `Energy2` Is currently typedef'ed to double but may in the future be using the SIUnits (?) package



# Status

THEPEG exists and is working. Snapshots of the current development code is available from <http://www.thep.lu.se/ThePEG>. Version 1.0 $\alpha$  will be released this weekend.

PYTHIA7 is now based on THEPEG. It exists and is working. Snapshots of the current development code is available from <http://www.thep.lu.se/Pythia7>. Version 1.0 $\alpha$  will be released this weekend.

PYTHIA7/THEPEG includes some basic  $2 \rightarrow 2$  matrix elements, a couple of PDF parameterizations, remnant handling, initial- and final-state parton showers, Lund string fragmentation and flat n-body particle decays.



# Future Plans

- PYTHIA7: Rework fragmentation to include junction strings.
- PYTHIA7: Multiple Interactions<sup>c</sup>.
- PYTHIA7: Proper particle decays.
- PYTHIA7: All the rest...
- ARIADNE: Dipole shower.
- ARIADNE: LDC model with multiple interactions<sup>c</sup>.

---

<sup>c</sup>May imply changes to the way sub-processes are handled



