LUND
UNIVERSITY

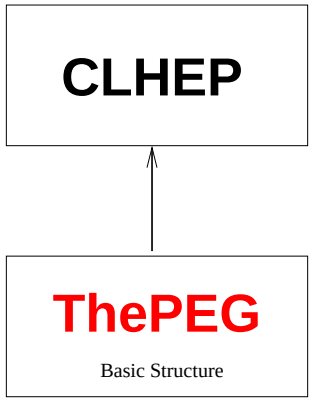
ThePEG/Pythia7 Tutorial

- Introduction
- Overview
- Installation
- How to write a simple (Analysis) handler
- How to write a matrix element handler
- Questions and discussion

CERN
2003.07.22
Leif Lönnblad

What is ThePEG

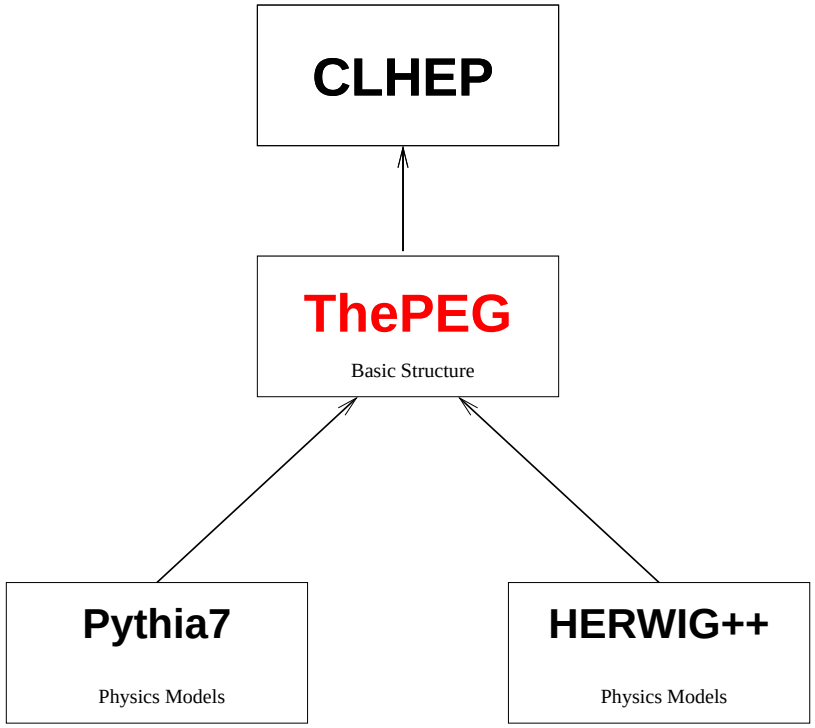
ThePEG consists of the parts of Pythia7 which were not specific to the Pythia physics models. It provides a general structure for implementing models for event generation.



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.

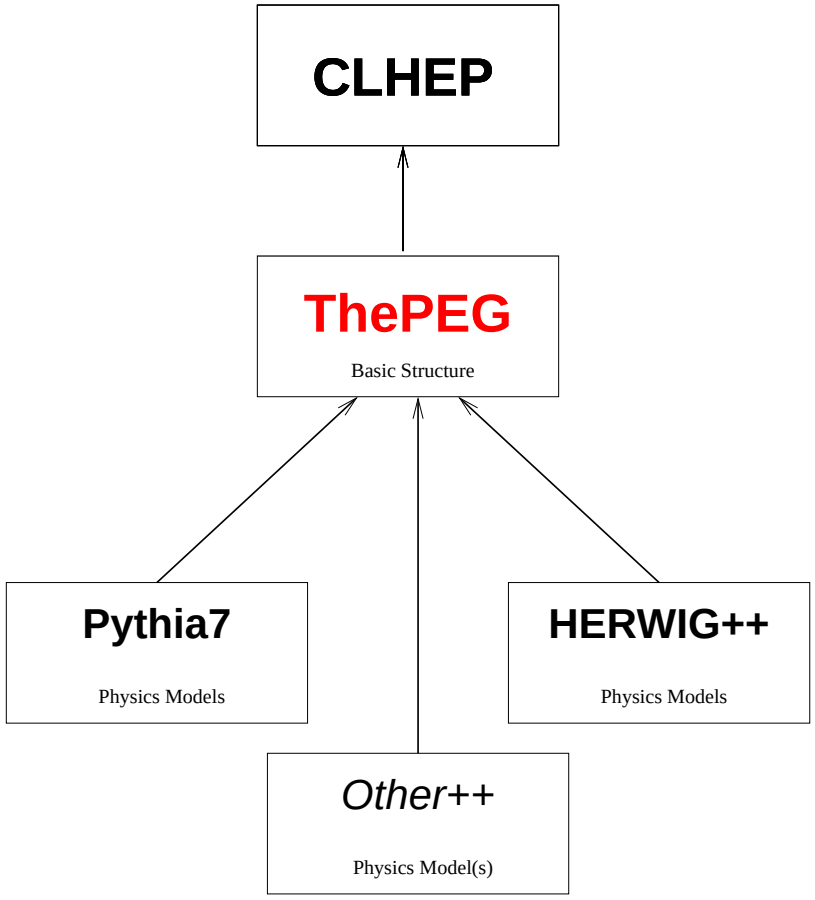


What is ThePEG

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...



ThePEG is a uniform interface between Event Generator Users and Event Generator Authors

- Authors Implement their physics models in ThePEG handlers (modules/plugins).
- Users link together the handlers and and generate events.

The components of ThePEG

- Basic infrastructure: Smart pointers, extended type information, object persistency, Exceptions, Dynamic loading, ...
- Kinematics: Extra utilities on top of CLHEP, 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 and the user.
- 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. Parameters and switches are registered with the `Repository` of ThePEG.

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.

  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.

- 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 [a] which can be sent to analysis or to detector simulation.

[a] `ThePEG::Events` which can be translated into `HepMC::GenEvents`

Installation

Make sure you are running Linux and have gcc-3.3 and CLHEP-1.8.

```
% wget http://www.thep.lu.se/ThePEG/ThePEG-1.0a.tgz
% wget http://www.thep.lu.se/Pythia7/Pythia7-1.0a.tgz
% tar xzf ThePEG-1.0a.tgz
% tar xzf Pythia7-1.0a.tgz
% cd ThePEG-1.0a
% setenv CLHEPPATH "/usr/local"
% setenv CLHEPINCLUDE "/usr/local/include"
% setenv CLHEPLIB "CLHEP-g++.1.8.0.0"
% ./configure --prefix=/usr/local
% make install
```

This assumes that CLHEP includes are in `/usr/local/include/CLHEP` and the CLHEP library is in `/usr/local/lib/libCLHEP-g++.1.8.0.0.a`

`make install` will build ThePEG and Pythia and run a few test programs before they are actually installed[a]

Header files will be in `/usr/local/include/ThePEG` and `/usr/local/include/Pythia7`. Shared libraries (and a few other files) will be in `/usr/local/lib/ThePEG/1.0a` and the setup and run programs will be in `/usr/local/bin`.

The test program for Pythia7 will run the setup program with the default Pythia7 repository and make an `EventGenerator` object which is written to disk. The `EventGenerator` is then read in and run.

[a] Due to a small bug this may fail. In that case try to rename the `libCLHEP-g++.1.8.0.0.so` in the `ThePEG/lib` directory to `libCLHEP.so` and `make install` again.

Writing a simple Handler class

Copy `/usr/local/lib/ThePEG/1.0a/Makefile` to your working directory.

Load `/usr/local/lib/ThePEG/1.0a/ThePEG.el` in emacs.

Do `M-x ThePEG-AnalysisHandler-class-files`. This will create `.h`, `.icc` and `.cc` files for your class with skeletons to fill in.

Add your files to the makefile and do `make Multiplicity.so`.

Objects of your class can now be added to the repository by the setup program.

The example class wass called `Multiplicity` and was in the namespace `Myclasses`. This is how to use it with PYTHIA7:

```
% setupThePEG -r Pythia7Defaults.rpo
ThePEG> mkdir /MyAnalysis
ThePEG> cd MyAnalysis
ThePEG> library Multiplicity.so
ThePEG> create /MyClasses/Multiplicity Mult
ThePEG> cd /Pythia7/Generators
ThePEG> insert StdLEP:AnalysisHandlers[0] /MyAnalysis/Mult
ThePEG> set StdLEP:NumberOfEvents 10000
ThePEG> saverun LEPTest StdLEP
ThePEG> [ctrl-d]
% runThePEG LEPTest.run
```

`[t][c]{Blah}Ptr` are smart pointers to objects inheriting from the `ReferenceCounted` base class. All relevant classes in ThePEG are reference counted.

Some common classes have abbreviated pointer typedefs.

`PPtr` is a smart pointer to a `Particle` object.
`cPPtr` is a `const` smart pointer to a `Particle`
`tPPtr` is a normal (stupid) pointer to a `Particle`
`tcPPtr` is a normal `const` pointer to a `Particle`

Instead of doing

```
Particle * pp = new Particle(arg);
...
delete pp;
```

you do

```
PPtr pp = new_ptr(Particle(arg));
...
```

the object will be automatically deleted when no more (smart) pointers are referring to it.

Object Persistency

Needed to be able to write objects to a file and read them back in again and preserving their relationships.

To accomplish this, the type informtion in C++ has to be expanded.

For each class there must exist one static object of the templated `ClassDescription` class. In addition, the templated `BaseClassTrait` must be specialized for each class to specify the base class, and the templated `ClassTrait` class must be specialized to give the class name, and the name of the shared library where the code for the class resides.

An object can then be written to a `PersistentOStream` and read back again from a `PersistentIStream`.

If a class has member variables which should keep their value after write/read, it must implement

```
void persistentOutput(PersistentOStream &) const;
void persistentInput(PersistentIStream &, int);
```

where each variable are written (with $<<$) and read (with $>>$) to the corresponding stream.

Interfaced classes

Classes which are to be handled by the repository in the setup program must inherit from the `Interfaced` class.

There are then a number of methods which may be implemented:

```
virtual IBPtr clone() const;
virtual void doupdate() throw(UpdateException);
virtual void doinit() throw(InitException);
virtual void doinitrun();
virtual void dofinish();
virtual void rebind(const TranslationMap & trans);
virtual IVector getReferences();
```

Process Generation

ThePEG goes beyond the Les Houches Accord. Matrix Element objects are used inside ThePEG to generate sub-processes. [a]

Briefly the matrix element class must be able to return the matrix element squared for a given kinematic configuration of incoming and outgoing partons; to construct the kinematics of the partons given a set of numbers $\vec{r}$ in the range $]0,1[$; to give $d\hat{\sigma}/d\vec{r}$ and to specify colour lines for the partons.

[a] The `PartialCollisionHandler` can be used to read in Les Houches files and cascade, fragment and decay.

Here are some virtual functions which need to be implemented.

```
virtual void getDiagrams() const;
```

Tell the base class which diagrams can be generated. Diagrams are represented by objects of classes inheriting from `DiagramBase`. All that THEPEG cares about is incoming and outgoing partons. `Tree2toNDiagram` is provided by THEPEG and should be suitable for most cases.

```
virtual int nDim() const;
virtual bool generateKinematics(const double * r);
```

How many dimensions of $\vec{r}$ are needed for generating the kinematics.

For a previously specified $\hat{s}$ and the given $\vec{r}$, generate the kinematics.

```
virtual double me2() const;
virtual CrossSection dSigHatDR() const;
```

Return the squared matrix element and $d\hat{\sigma}/d\vec{r}$ for previously generated kinematics.

```
virtual Energy2 scale() const;
```

Return the scale associated with the previously generated kinematics.

```
virtual DiagramIndex diagram(const DiagramVector &) const;
virtual const ColourLines & selectColourGeometry(tcDiagPtr diag) cons
```

Select a diagram and a colour geometry for the previously generated kinematics.

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).

```
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).

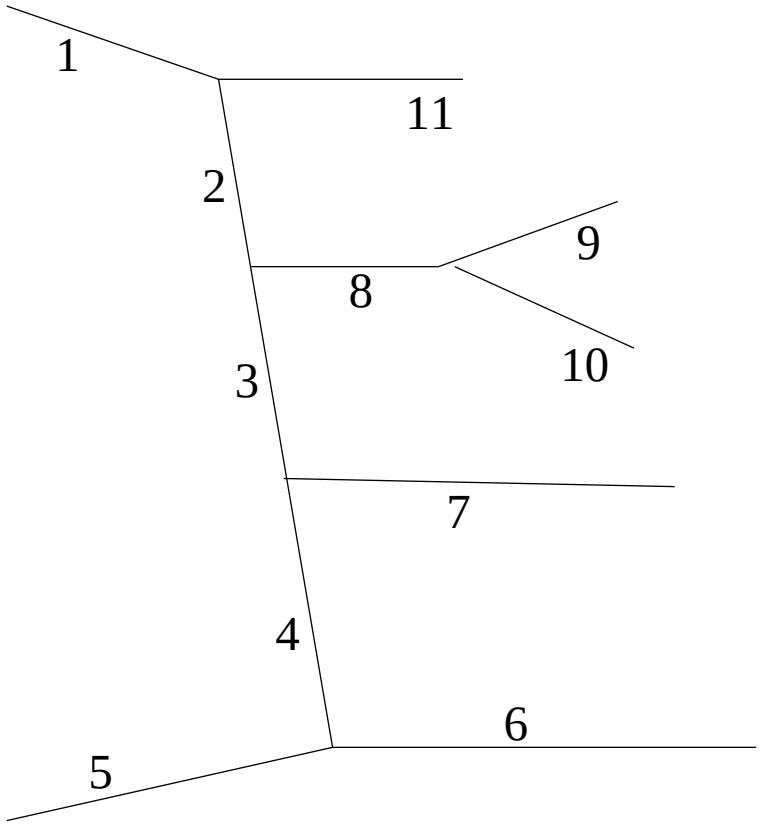
- `PDPtr` (smart) pointer to a `ParticeData` object
- `PDVector` vector of pointers to `ParticeData` objects
- `Energy2` Is currently `typedef`'ed to `double` but may in the future be using the SIUnits (?) package

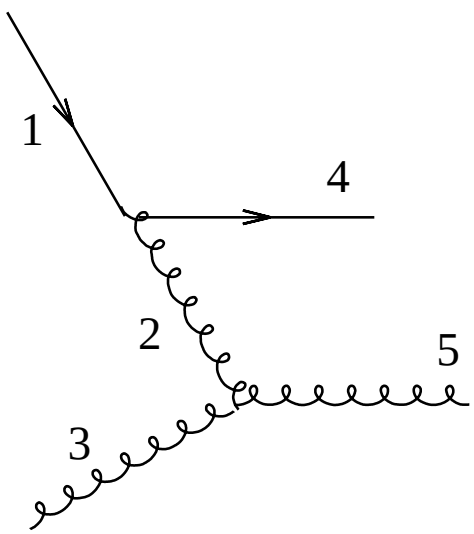
A `SamplerBase` object is responsible for the integration and sampling of

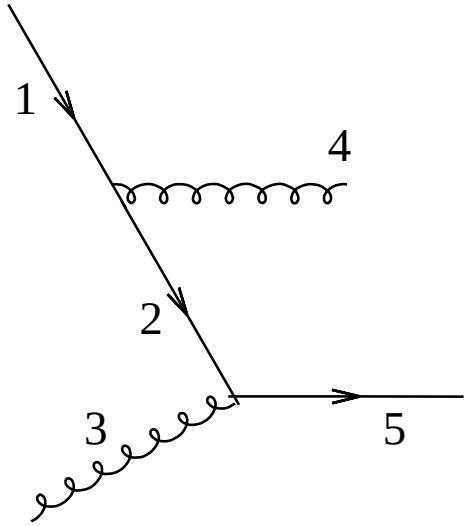
$$\int_0^1 \prod_i dr_i \;\; f_a(l_a(r_0)) \frac{dl_a}{dr_0} f_b(l_b(r_1)) \frac{dl_b}{dr_1} \frac{d\hat{\sigma}(s/\exp(l_a+l_b), r_2, \ldots, r_n)}{dr_2 \ldots dr_n}$$

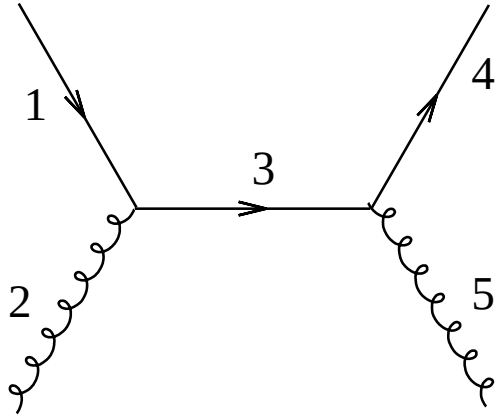
$d\hat{\sigma}/d\vec{r}$ does not need to be flat in $\vec{r}$, but the matrix element object should try to flatten the phase space as much as it can in `generateKinematics(const double * r)`. Also the `PDFBase` object may flatten its phase space if needed.

An object of the `Tree2toNDiagram` class which has five space-like lines and a number of time-like lines.




1
4
2
5
3


1
4
2
3
5


1
4
3
2
5