



LUND
UNIVERSITY



PYTHIA7 (PYTHIA \rightarrow C++)

- Introduction
- Overview
- Parton Densities
- Matrix Elements
- Future Plans

Fermilab
2001.04.16
Leif Lönnblad

What is PYTHIA7

A project to completely rewrite the **Lund** family of event generators (PYTHIA, ARIADNE, ...) in C++.

The main target is LHC physics, but the PYTHIA7 can be used for simulation of any high-energy particle collisions. (Heavy-ions are not a priority)

Not only the **Lund Model**. It should provide a general structure for implementing models for event generation.

The aim is to define the standard generator platform for HEP in general and LHC in particular.



Why?



The structure of event generation

The standard way of generating an event is to start by selecting a hard sub-process using parton density functions and $2 \rightarrow n$ partonic matrix elements, and then add partonic showers, hadronization and decay of unstable particles.

In general, there is sequence of steps to go through, typically ordered in scale or virtuality of the physical processes involved. Each step corresponds to a physical model for a factorizable part of the event generation.

But it can become very complicated when adding multiple (semi-)hard interactions, overlaid events, colour reconnections, etc. . .



PYTHIA7 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.



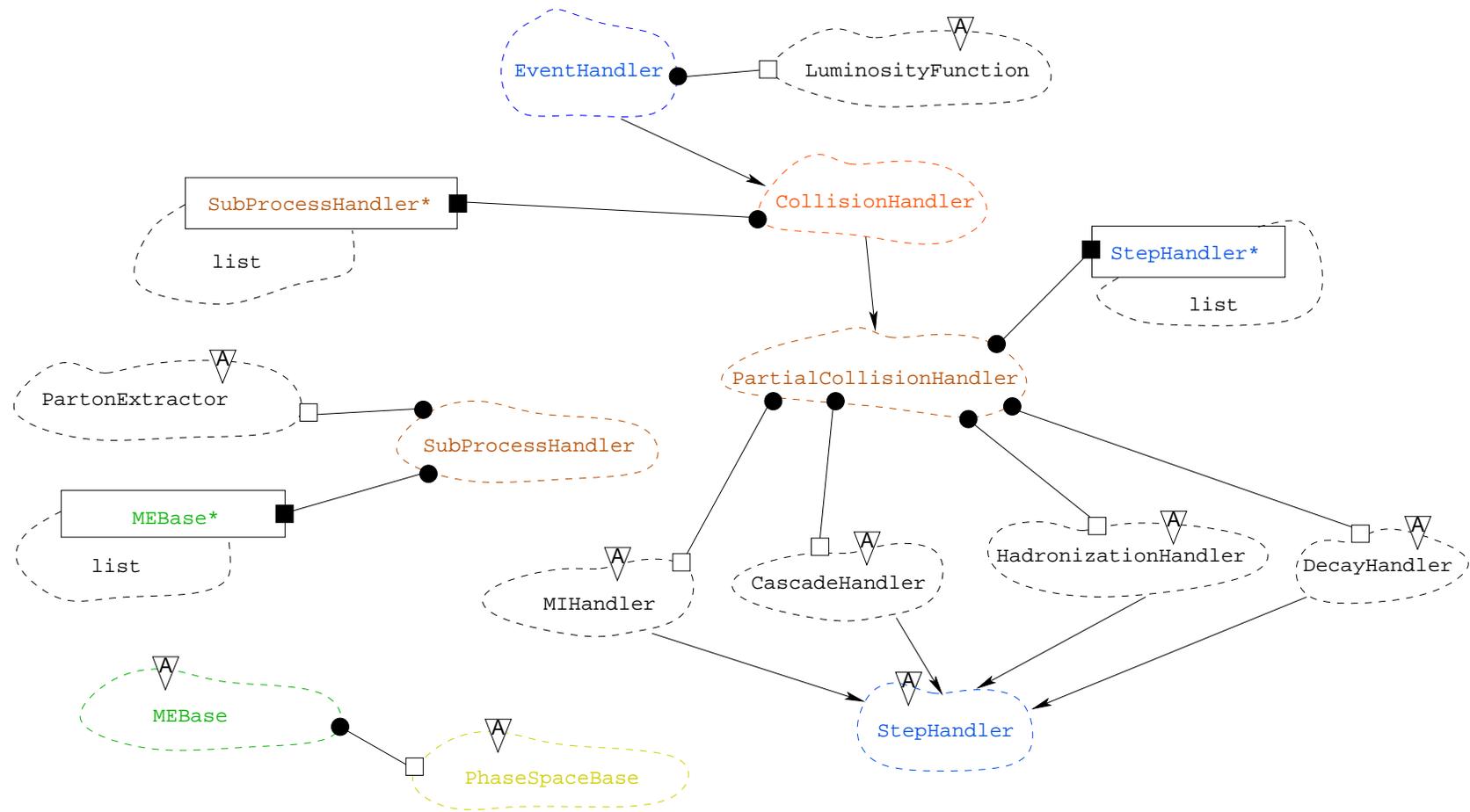
The structure of the generation process is extremely dynamic:

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.

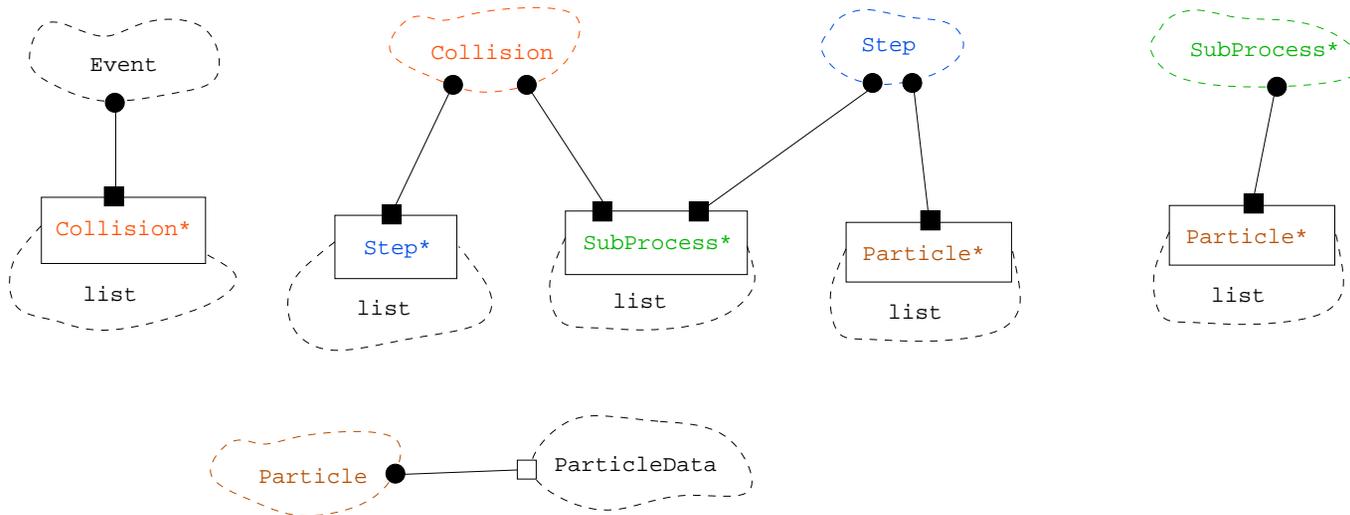
In addition, each handler can add steps in the generation chain or redo previous steps depending on the history of each event.



Class structure of *handlers*



Class structure of an Event



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

```
Event ev;
// generate the event...
vector<ParticlePtr> particles;
ev.select(back_inserter(particles), SelectFinalState());
```



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 if needed.

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



Running PYTHIA7

The end-user will use a setup program to be able to pick objects corresponding to different physics models to build up an **EventGenerator** which then can be run interactively or off-line. This is typically done by choosing between a multitude of pre-defined generators, modifying only a few steps.

The same setup program is used to modify parameters and options of the selected models and, optionally, to specify the analysis to be done on the generated events.

The **EventGenerator** can be saved to a file to be read into a special slave program for running or into e.g. Geant4.



The **Repository** is the central part of the setup phase. It handles a structured list of all available objects and allows the user to manipulate them.

A flashy Graphical User Interface should be built on top of this **Repository**. Currently there is only a rudimentary command-line interpreter:

```
cd /Defaults/EEGenerators
set SimpleLEPHandler:HadronizationHandler stdLundFragHandler
set stdLundFragHandler:FlavourGenerator:BaryonMode 1
set stdLundFragHandler:ZGenerator:a 0.24
set SimpleLEPGenerator:EventHandler SimpleLEPHandler
set SimpleLEPGenerator:NumberOfEvents 10000
saverun SimpleLEP SimpleLEPGenerator
```



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).



```
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 will in the future be using the `SIUnits` package



Implementing Matrix Elements

Matrix elements are implemented in two different class hierarchies inheriting from `MEBase` and `PhaseSpaceBase`. `MEBase` implements the actual squared matrix element function, while `PhaseSpaceBase` is responsible for generating phase space points according to this matrix element in an efficient way.

Which `PhaseSpaceBase` object is used by a particular `MEBase` can be modified through the user interface.

The communication between the objects is handled with `TreeDiagram` objects with which the `MEBase` describes what kind of diagrams are included, mainly to reveal the pole structure of the matrix elements.

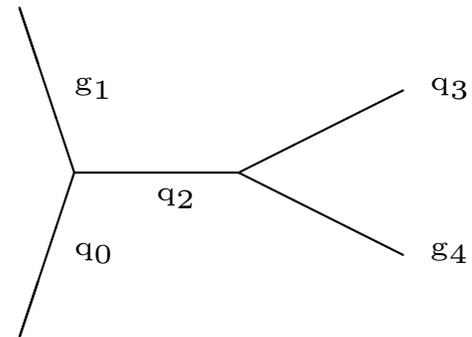
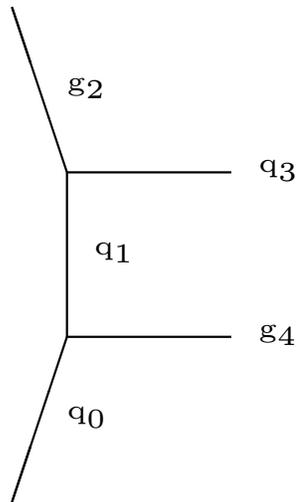
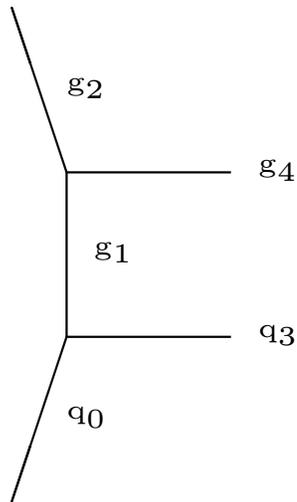


A matrix element class must implement the abstract

```
DiagramVector includedDiagrams() const;
```

defined in **MEBase** to return the **TreeDiagrams** which are implemented.

```
MEQG2QG::DiagramVector MEQG2QG::includedDiagrams() const {
DiagramVector ret;
tcPDPtr g = getParticleData(ParticleID::g);
for ( int i = 1; i <= maxFlavour(); ++i ) {
tcPDPtr q = getParticleData(i);
ret.push_back((TreeDiagram(3), q, g, g, 0, q, 1, g, -1));
ret.push_back((TreeDiagram(3), q, q, g, 1, q, 0, g, -2));
ret.push_back((TreeDiagram(2), q, g, 0, q, 2, q, 2, g, -3));
q = getParticleData(-i);
ret.push_back((TreeDiagram(3), q, g, g, 0, q, 1, g, -1));
ret.push_back((TreeDiagram(3), q, q, g, 1, q, 0, g, -2));
ret.push_back((TreeDiagram(2), q, g, 0, q, 2, q, 2, g, -3));
}
return ret;
}
```



Other abstract methods in `MEBase`:

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

Return the squared matrix element using the Lorentz momenta and types of the incoming and outgoing partons available through simple member functions.

```
virtual Selector<ColourGeometry>  
  colourGeometries(const TreeDiagram & diag) const;
```

Return a `Selector` where possible `ColourGeometries` at the current phase space point and the selected diagram are rated with their relative weights (un-physical but necessary). Numbering the incoming partons 1 and 2, and the outgoing $3 \dots n + 2$ in the order they were specified in the `TreeDiagram`

```
ColourGeometry(3, 1, 4, 2);
```

specifies that the colour flows from $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$.



Common member functions

There are a number of member functions and *traits* classes which need to be implemented for each new class in PYTHIA7. Some of these are needed since the C++ standard doesn't include useful RTTI and object persistency, others are needed by the **Repository**. Many of these can (and should in the future) be implemented automatically.



Common member functions

There are a number of member functions and *traits* classes which need to be implemented for each new class in PYTHIA7. Some of these are needed since the C++ standard doesn't include useful RTTI and object persistency, others are needed by the **Repository**. Many of these can (and should in the future) be implemented automatically.

The following is needed for the extended RTTI:

- Exactly one static object of type `ClassDescription<Foo>` must be defined.
- The templated class `ClassTraits<Foo>` may be specialized if needed
- The templated class `BaseClassTraits<Foo,BaseI>` should be specialized for each base class



For the object persistency scheme in PYTHIA7, the following methods need to be implemented (if nothing else is specified in the `ClassTraits<Foo>` class):

```
void Foo::persistentOutput(PersistentOStream & os) const {
    os << member << another << newmember;
}
void Foo::persistentInput(PersistentIStream & is, int version) {
    is >> member >> another;
    if ( version >= 1 ) is >> newmember;
}
```



For the object persistency scheme in PYTHIA7, the following methods need to be implemented (if nothing else is specified in the `ClassTraits<Foo>` class):

```
void Foo::persistentOutput(PersistentOStream & os) const {
    os << member << another << newmember;
}
void Foo::persistentInput(PersistentIStream & is, int version) {
    is >> member >> another;
    if ( version >= 1 ) is >> newmember;
}
```

To work with the `Repository` the following methods must be implemented:

```
virtual IBPtr clone() const;
```

should return an exact copy.

```
static Init();
```

Could be empty, but is typically used to define static `Parameter`, `Switch` and `Reference` objects which can be used by the `Repository` to manipulate objects of this class.



PYTHIA7 makes extensive use of new features in the C++ standard, e.g. templates exceptions, standard (container) library, RTTI, namespaces...

Today it takes forever to compile and the executables are huge - but this situation will improve as compilers handling of templates matures.

The exception mechanism is perfect for the veto-algorithm and is used extensively, but it is currently extremely slow. This will also improve as compilers mature (I hope).



Future Plans

PYTHIA7 exists today only as a proof-of-concept version available from <http://www.thep.lu.se/Pythia7>

It includes some basic $2 \rightarrow 2$ matrix elements, a couple of PDF parameterizations, remnant handling and Lund string fragmentation.

Main priorities:

- Parton Showers
- Multiple interactions
- Particle Decays
- Clean up the code



Time plan

Well suited for Tevatron Run-III!



Time plan

NOT Well suited for Tevatron Run-III!



Time plan

NOT Well suited for Tevatron Run-III!

- A new pre-release after the summer with the first HERWIG stuff.
- A usable generator in 2002. (with basic PYTHIA and HERWIG stuff)
- The Standard Generator at the start-up of LHC.

Help will be appreciated. There are a lot of well defined tasks which can be outsourced. Unfortunately there is today a fairly high threshold before you can start working with it.

