StdHepC++

L. Garren

Fermi National Accelerator Laboratory, USA

Abstract

StdHepC++[1] is a proposed standard CLHEP Monte Carlo event class library which will allow users a common interface to Monte Carlo event generators. The HEPEVT common block, together with the StdHep Fortran interface, and the PDG particle numbering scheme provide the current Fortran standard. The new classes will provide an object oriented interface to the current functionality, as well as allowing for multiple collisions, tracing of parents and daughters, and I/O in several standard formats.

It is hoped that this class can be used directly by the C++ event generators now being coded. At minimum, methods will be available to fill the StdHepC++ classes from the information in Monte Carlo event generators. The intent is to use these classes with detector simulators such as Geant4 and other analysis routines.

Keywords: stdhep

1 Introduction

As is well known, every Monte Carlo generator has its own interface and particle definitions. Yet users need to process information from various generators in a standard way. This problem was previously solved by use of the HEPEVT[2, 3] common block and the PDG standard particle numbering scheme[4].

As physicists move from Fortran to C++, it is necessary to provide C++ tools for analysis. Also, a number of generators are being written in or converted to C++[5]. Further, events and particles can be naturally described as objects.

We propose a standard event generator interface for C++: StdHepC++[1]. This interface will be in CLHEP[6]. The intent is to create modules which can be used directly by Monte Carlo generators so that specialized StdHep translation routines will no longer be needed.

As a first step, an interface to the existing Fortran HEPEVT common block has been created. The StdHepC++ class library will implement all functions currently in the Fortran StdHep library. Some of the current functionality is: translation between Monte Carlo generator output and StdHep; translation of particle ID codes to a consistent set; translation from StdHep to simulator input (e.g. GEANT); charge, name, and quark content of a particle; event specific properties, including lists of descendants and ancestors; and platform independent I/O using XDR via the implementation used in the MCFast[7] simulation package.

2 The Code

2.1 Classes

StdHepC++ uses 5 main classes in the StdHep namespace:

- Particle
- ParticleData
- Collision
- Event
- Run

StdHep::ParticleData contains standard particle information for each type of particle (e.g. π^0, η', etc.): particle ID number[4], particle name, charge, mass, width, lifetime, spin, and isospin. This is the standard Particle Data Group information and can be referenced via particle ID. Ideally, this would also contain the particle decay table, however a general purpose access to such information is outside the scope of this project.

StdHep::Particle contains the volatile particle information: Particle ID, status code, mother, second mother, an index to the first daughter, an index to the last daughter, color, momentum CLHEP Lorentz vector, generated mass, helicity, creation vertex CLHEP Lorentz vector, and decay vertex CLHEP Lorentz vector. The mother, second mother, and indices to first and last daughter are deliberately designed to be compatible with the HEPEVT common block. StdHep::Particle also contains a method to access the appropriate StdHep::ParticleData.

StdHep::Collision contains a vector of pointers to particles, a collision number, the number of particles, and the input I/O stream. A collision is a single interaction.

Because there may be several interactions in one beam crossing, an event is a collection of collisions. StdHep::Event has a vector of pointers to collisions, an event number, and the number of collisions.

StdHep::Run contains the number of events to generate, the number of events actually generated, and event independent information, such as the number of events written to I/O, the nominal center of mass energy, the cross-section, and random number seeds. This class contains methods for efficient I/O of events or portions of events.

2.2 Methods

Basic methods include class constructors, copy constructors, and accessors and modifiers for every element of the classes. There are also methods to return various collections of descendents and ancestors, e.g. StdHep::Descendants, StdHep::StableDescendants, and StdHep::ChargedStableDescendants.

The non-trivial methods in StdHepC++ center on platform independent I/O. The StdHep namespace has InitReadXDR, OpenReadXDR, ReadXDR, InitWriteXDR, OpenWriteXDR, WriteXDR, WriteEventXDR, and WriteEndXDR. The ReadXDR methods accept data and build the StdHep classes. Simulations and user analyses use these methods to work with the data from arbitrary generators. The WriteXDR methods translate the class information into XDR C information. These methods are used to output information in a portable format. StdHepC++ uses the MCFio XDR implementation, which can be found in both MCFast and StdHep.

The I/O methods will also address other formats used in important HEP tools. The I/O approach adopted by LHC++[8] will have direct support. Also, designs for rapid I/O of subsets of Particle properties will be developed in cooperation with the ROOT[9] team.

3 Directed Acyclic Graphs

We propose to put the Particles comprising a Collision into a directed acyclic graph (DAG) instead of a vector. A DAG is a templated class which knows the parent/child relations, so the Particles can be envisioned as associated with points on a graph and these points can be connected to one another by arcs representing the relationships. This provides a natural method for determining parent/child relationships in the particle list, yet separates the relationship data from the intrinsic data describing

each Particle. Once the DAG is implemented, the mother and daughter information can be removed from the Particle class.

The DAG can contain arbitrary numbers of points, children, parents, and roots. Roots are the points with no parents. Note that children can have multiple parents, and of course, parents can have multiple children. However, there cannot be more than one arc between any two given points. Also, there cannot at any time be a closed circle in the graph. Methods which add Particles or assign relationships will enforce this.

A full description of this proposal is available at
`http://www-pat.fnal.gov/stdhep/c++/dag.txt`.

4 Conclusion

There is a strong need for a C++ standard Monte Carlo generator interface. StdHepC++ is a natural object-oriented implementation of such an interface. At present, we have working examples which integrate StdHepC++ with the Fortran versions of Herwig, Pythia, and Isajet.

References

1 StdHepC++: `http://www-pat.fnal.gov/stdhep/c++/`.
2 T. Sjöstrand *et al.*, in "Z physics at LEP1", CERN 89-08, vol. 3, p.327.
3 T. Sjöstrand, "Interfacing four-fermion generators with QCD generators", Workshop on Physics at LEP2, (Jan. to Oct. 1995).
4 Particle Data Group: C. Caso *et al.*, The European Physical Journal **C3** (1998) 180
`http://www-pdg.lbl.gov/mc_particle_id_contents.html`.
5 Pythia7: `http://www.thep.lu.se/tf2/staff/leif/Pythia7/Welcome.html`.
6 CLHEP: `http://wwwinfo.cern.ch/asd/lhc++/clhep/`.
7 MCFast: `http://www-pat.fnal.gov/mcfast.html`.
8 LHC++: `http://wwwinfo.cern.ch/asd/lhc++/`.
9 ROOT: `http://root.cern.ch/`.