```
// ----------------------------------------------------------
// StdCollision.hh
// ----------------------------------------------------------

#ifndef STDCOLLISION_HH
#define STDCOLLISION_HH

#include <vector>
#include "StdParticle.hh"

namespace StdHep {

class Collision {

public:

  // ---  birth/death:
  //
  Collision( int CollisionEventNumber, int IOStream,
             int ntries, vector<Particle*> & pvect );
  Collision( );
  Collision( const Collision & orig );
  Collision& operator=( const Collision & rhs );
  ~Collision();

  // ---  accessors:
  //
  int CollisionEventNumber()  const
                         { return itsCollisionEventNumber; }
  int IOStream()              const  { return itsIOStream; }
  int Entries()               const  { return itsEntries; }
  int ListSize()              const
                         { return itsParticleList.size(); }

  Particle  CollParticle( int i )   const
                         {  return *(itsParticleList[i]); }

  // ---  mutators:
  //
  void  setCollisionEventNumber ( int evn )
                          { itsCollisionEventNumber = evn; }
  void  setIOStream             ( int str )
                          { itsIOStream = str; }
  void  setEntries              ( int ntries )
                          { itsEntries = ntries; }
  void  appendParticle          ( Particle* p );

private:

};      // Collision

}
#endif // STDCOLLISION_HH
```

```
// --------------------------------------------------------
//
// ReadParticleDataMap.hh
//
// collect all the signatures for input data methods here
//
// --------------------------------------------------------

#ifndef READPARTICLEDATA_HH
#define READPARTICLEDATA_HH

#include "SimParticleMap.hh"

// read the PDG list
int ReadPDGData( const char *filename, SimParticleMap & sd );

#endif // READPARTICLEDATA_HH
```

```
// ------------------------------------------------------------
//
// SimParticleData.hh
//
// ------------------------------------------------------------
#ifndef SIMPARTICLEDATA_HH
#define SIMPARTICLEDATA_HH

#include <string>
#include <vector>

#include "SimParticleID.hh"

class SimParticleData  {

public:

  // ---  birth/death:
  //
  SimParticleData( const std::string & ParticleName, int ID,
                   int Charge, int Spin, int Isospin,
                   double Mass, double MassSigma,
                   double Width, double WidthSigma,
                   double Lifetime, double LifetimeSigma );
  SimParticleData( const std::string & ParticleName, int ID,
                  double Charge, double Spin, double Isospin,
                  double Mass,  double MassSigma,
                  double Width, double WidthSigma,
                  double Lifetime, double LifetimeSigma);
  SimParticleData( const SimParticleData & orig );
  SimParticleData& operator=( const SimParticleData & );
  ~SimParticleData();

  // ---  accessors:
  //
  const std::string & ParticleName()    const
                             { return itsParticleName; }
  SimParticleID  ID() const { return itsID; }
  double    Charge() const { return double(itsCharge)/3.0; }
  double    Spin()   const { return double(itsSpin-1)/2.0; }
  double Isospin() const { return double(itsIsospin-1)/2.0; }
  double   Mass()         const { return itsMass; }
  double   MassSigma()    const { return itsMassSigma; }
  double   Width()        const { return itsWidth; }
  double   WidthSigma()   const { return itsWidthSigma; }
  double   Lifetime()     const { return itsLifetime; }
  double   LifetimeSigma() const { return itsLifetimeSigma; }
  int      NumDecays()   const { return itsDecayList.size(); }
  SimParticleDecay  getDecay( int i ) const
                            { return itsDecayList[i]; }
```

```
  // ---  mutators:
  //
  void  setID( int ID )       { itsID = SimParticleID( ID ); }
  void  setCharge( int Charge )   { itsCharge = Charge; }
  void  setCharge( double Charge )
                         { itsCharge = int(3.0*Charge); }
  void  setSpin( int spin )   { itsSpin = spin; }
  void  setSpin( double spin )
                         { itsSpin = int(2.0*spin + 1.0); }
  void  setIsospin( int isospin ) { itsIsospin = isospin; }
  void  setIsospin( double isospin )
                     { itsIsospin = int(2.0*isospin + 1.0); }
 void setMass( double mass )        { itsMass = mass; }
 void setMassSigma( double mass )   { itsMassSigma = mass; }
 void setWidth( double width )      { itsWidth = width; }
 void setWidthSigma( double width ){ itsWidthSigma = width; }
 void setLifetime( double life )    { itsLifetime = life; }
 void setLifetimeSigma( double ls ) { itsLifetimeSigma=ls; }
 void setMassandSigma( double val, double err )
                      { itsMass = val; itsMassSigma = err; }
 void setWidthandSigma( double val, double err )
                      { itsWidth = val; itsWidthSigma = err; }
 void appendDecay( SimParticleDecay );

  // create particle pair
  void ParticleDataPair( const std::string & ParticleName,
                 int ID, int Charge, int Spin, int Isospin,
                 double Mass, double MassSigma,
                 double Width, double WidthSigma,
                double Lifetime, double LifetimeSigma );
  void ParticleDataPair( const std::string & ParticleName,
         int ID, double Charge, double Spin, double Isospin,
         double Mass,  double MassSigma,
         double Width, double WidthSigma,
         double Lifetime, double LifetimeSigma);

private:

  // null constructor not allowed
  SimParticleData( );

  // What are the implications of including Color?
  // Do you need 3 of each quark and antiquark?

  vector<SimParticleDecay> itsDecayList;
  // other private items not listed

};  // SimParticleData

#endif // SIMPARTICLEDATA_HH
```

```
// -----------------------------------------------------
//
// SimParticleDecay.hh
//
// -----------------------------------------------------

#ifndef SIMPARTICLEDECAY_HH
#define SIMPARTICLEDECAY_HH

#include <string>
#include <vector>

#include "SimParticleID.hh"

class SimParticleDecay  {

public:

  // ---  birth/death:
  //
  SimParticleDecay( int type, double rate, double hel,
                    vector<SimParticleID> decayl );
  SimParticleDecay( );
  SimParticleDecay( const SimParticleDecay & orig );
  SimParticleDecay& operator=( const SimParticleDecay & );
  ~SimParticleDecay();

  // ---  accessors:
  //
  int    DecayType() const  {  return itsDecayType; }
  double Rate()      const  {  return itsRate; }
  double Helicity()  const  {  return itsHelicity; }
  int    DecaySize() const { return itsParticleList.size(); }

  // ---  mutators:
  //
  void  setDecayType( int dt )     {  itsDecayType = dt; }
  void  setRate( double rate )     {  itsRate = rate; }
  void  setHelicity( double hel ) {  itsHelicity = hel; }
  void  appendParticle( SimParticleID id );

private:
  // other private items not listed
  vector<SimParticleID> itsParticleList;

};  // SimParticleDecay

#endif // SIMPARTICLEDECAY_HH
```

```
// ------------------------------------------------------------
//
// SimParticleID.hh
//
//  various utilities to extract information from the PID
//
// ------------------------------------------------------------

#ifndef SIMPARTICLEID_HH
#define SIMPARTICLEID_HH

class SimParticleID {

public:

  // ---  birth/death:
  //
  SimParticleID( int pid);
  SimParticleID( );
  SimParticleID( const SimParticleID & orig );
  SimParticleID& operator=( const SimParticleID & );
  ~SimParticleID();

  // ---  accessors:
  //
  int getPID() const { return itsPID; }
  const int  ChargeInt( );
  const double ChargeReal( );
  const int  GetJSpin( );
  const int  GerLSpin( );
  const void GetQuarks( int& q1, int& q2, int& q3 );
  const void GetChargedQuarks( int& q1, int& q2, int& q3 );
  const void SplitPID( int& js, int& q1, int& q2, int& q3,
                       int& ls, int& nr, int& nx,
                       int& susyID, int& chg, int& extra);

private:

  int          itsPID;

};  // SimParticleID

#endif // SIMPARTICLEID_HH
```

```
// -----------------------------------------------------------
//
// SimParticleMap.hh
//
// -----------------------------------------------------------

#ifndef SIMPARTICLEMAP_HH
#define SIMPARTICLEMAP_HH

#include <string>
#include <map>
#include "SimParticleData.hh"

class SimParticleMap  {

public:

  // ---  birth/death:
  //
 SimParticleMap( map<SimParticleID, SimParticleData*> & pd );
 SimParticleMap( );
 ~SimParticleMap();

  // ---  accessors:
  //
  int PDSize()    const { return itsParticleDataMap.size(); }
  SimParticleData* getParticleData( SimParticleID );
  int WriteParticleData( const char *filename );

  // ---  mutators:
  //
  void addParticleData   ( SimParticleData* p );
  // create antiparticle
  void AntiParticle( const SimParticleData & orig );

protected:

  // ---  birth/death:
  SimParticleMap( const SimParticleMap & orig );
  SimParticleMap& operator=( const SimParticleMap & );

private:

  bool IOwnParticles;
  map<SimParticleID, SimParticleData*> itsParticleDataMap;

};  // SimParticleMap

#endif // SIMPARTICLEMAP_HH
```

```
// ------------------------------------------------------------
// StdEvent.hh
// ------------------------------------------------------------
#ifndef STDEVENT_HH
#define STDEVENT_HH

#include <vector>

#include "StdCollision.hh"

namespace StdHep  {

class Event {

public:

   // ---  birth/death:
   //
   Event( int EventNumber, int NumberParticles,
          int NumberCollisions, vector<Collision*> & cvect );
   Event( );
   Event( const Event & orig );
   Event& operator=( const Event & );
   ~Event();

   // ---  accessors:
   //
  int        EventNumber() const { return itsEventNumber; }
  int    NumberParticles() const { return itsNumberParticles; }
  int NumberCollisions() const { return itsNumberCollisions; }
  Particle   HEParticle( int i )    const   ;
  Collision HEPCollision( int i ) const
                         { return *(itsCollisionList[i]); }

   // ---  mutators:
   //
void setEventNumber ( int nmbr } { itsEventNumber = nmbr; }
void setNumberParticles (int n) { itsNumberParticles = n; }
void setNumberCollisions (int n) { itsNumberCollisions = n; }
void  appendCollision     ( Collision* c );

   // ---  event translators
   //
   int CopyToTemp( int istr );
   int CopyFromTemp( int istr );
   int FtoCXX( int istr );
   int CXXtoF( int istr );
   int CtoCXX( int istr );
   int CXXtoC( int istr );
   void ZeroHEPEVT( void );

};        // Event
}
#endif // STDEVENT_HH
```

```
// ----------------------------------------------------------
// StdEventUtil.hh
// ----------------------------------------------------------

#ifndef STDEVENTUTIL_HH
#define STDEVENTUTIL_HH

#include <string>
#include <fstream>

#include "StdEvent.hh"

namespace StdHep  {

 //
 // - I/O for the CLHEP test
 int DumpEvent ( std::ofstream & dfile,
                 StdHep::Event & evnt );
 int ReadDumpedEvent ( std::ifstream & dfile,
                       StdHep::Event & evnt );
 void PrintEvent ( std::ofstream & ofile,
                   StdHep::Event & evnt );

}        // StdHep

#endif // STDEVENTUTIL_HH
```

```
// ---------------------------------------------------
// StdFI.hh
// ---------------------------------------------------
//
// This is the Fortran interface to various common blocks
//
#ifndef STDFI_HH
#define STDFI_HH

struct hepevt;
struct hepev2;
struct hepev3;

namespace StdHep  {

class FIhepevt {

public:

  FIhepevt( hepevt * h) { hptr = h; }
  static hepevt & get( ) { return * FIhepevt::hptr; }

private:
  static hepevt * hptr;
};        // FIhepevt

class FIhepev2 {

public:

  FIhepev2( hepev2 * h) { hptr = h; }
  static hepev2 & get( ) { return * FIhepev2::hptr; }

private:
  static hepev2 * hptr;
};        // FIhepev2

class FIhepev3 {

public:

  FIhepev3( hepev3 * h) { hptr = h; }
  static hepev3 & get( ) { return * FIhepev3::hptr; }

private:
  static hepev3 * hptr;
};        // FIhepev3

}         // StdHep

#endif // STDFI_HH
```

```
// ------------------------------------------------------------
// StdParticle.hh
// ------------------------------------------------------------
#ifndef STDPARTICLE_HH
#define STDPARTICLE_HH

#include "CLHEP/Vector/ThreeVector.h"
#include "CLHEP/Vector/LorentzVector.h"

namespace StdHep {

class Particle {

public:

  // ---  birth/death:
  //
  Particle( int ParticleID, int StatusCode, int Mother,
            int SecondMother,int BeginDaughters,
            int EndDaughters, int Color, int CollisionNumber,
            HepLorentzVector Momentum, double Mass,
            double Helicity, HepLorentzVector CreationVertex,
            HepLorentzVector DecayVertex );
  Particle( );
  Particle( const Particle & orig );
  Particle& operator=( const Particle & );
  ~Particle();

  // ---  accessors:
  //
  int   ParticleID()      const { return itsParticleID; }
  int   StatusCode()      const { return itsStatusCode; }
  int   Mother()          const { return itsMother; }
  int   SecondMother()    const { return itsSecondMother; }
  int   BeginDaughters()  const { return itsBeginDaughters; }
  int   EndDaughters()    const { return itsEndDaughters; }
  int   Color()           const { return itsColor; }
  int   CollisionNumber() const { return itsCollisionNumber; }
  HepLorentzVector Momentum() const { return itsMomentum; }
  double      Mass()            const { return itsMass; }
  double      Helicity()        const { return itsHelicity; }
  HepLorentzVector CreationVertex() const
                        { return itsCreationVertex; }
  HepLorentzVector DecayVertex()    const
                        { return itsDecayVertex; }

  // ---  mutators:
  //
  void  setParticleID     ( int pid )      { itsParticleID =
pid; }
  void  setStatusCode     ( int status )   { itsStatusCode =
status; }
  void  setMother         ( int mth )      { itsMother = mth;
```

```
  // ---  mutators:
  //
  void setParticleID    ( int pid ) { itsParticleID = pid; }
  void setStatusCode    ( int st )  { itsStatusCode = st; }
  void setMother        ( int mth ) { itsMother = mth; }
  void setSecondMother ( int mth ) { itsSecondMother = mth; }
  void setBeginDaughters ( int d ) { itsBeginDaughters = d; }
  void setEndDaughters   ( int d ) { itsEndDaughters = d; }
  void setColor          ( int clr ) { itsColor = clr; }
  void setCollisionNumber ( int c ) { itsCollisionNumber=c; }
  void setMomentum       ( HepDouble px, HepDouble py,
                           HepDouble pz, HepDouble E );
  void setMass      ( double mass ) {  itsMass = mass; }
  void setHelicity ( double hel )  {  itsHelicity = hel; }
  void setCreationVertex ( HepDouble cx, HepDouble cy,
                           HepDouble cz, HepDouble ct );
  void setDecayVertex    ( HepDouble dx, HepDouble dy,
                           HepDouble dz, HepDouble dt );


};       // Particle

}
#endif // STDPARTICLE_HH
```

```
// ----------------------------------------------------
// StdRun.hh
// ----------------------------------------------------

//  The basic I/O unit is a run
//  The run contains methods to get the next event

#ifndef STDRUN_HH
#define STDRUN_HH

namespace StdHep  {

class Run {

public:


  // ---  accessors:
  //
  virtual int      RunNumber()        const = 0;
  virtual int      EventsGenerated()  const = 0;
  virtual int      EventsWritten()    const = 0;

  // ---  mutators:
  //
  virtual void  setRunNumber         ( int runno )  = 0;
  virtual void  setEventsGenerated   ( int ngen )   = 0;
  virtual void  setEventsWritten     ( int nwrt )   = 0;

  // --- event access should be in this class
  //
  // here are some examples
  // int InitRead ( char*filename, int& ntries, int& istr );
  // int OpenRead ( char*filename, int& ntries, int& istr );
  // int Read   ( int& ilbl, int istr, StdHep::Event* & evt );
  // int InitWrite (char *filename, char *title, int ntries,
  //                 int& ist);
  // int Write     (int ilbl, int ist, StdHep::Event* evnt );
  // int WriteEnd (int ist);

  // --- version tracking
  void PrintVersion( void);

};       // Run

}        // StdHep

#endif // STDRUN_HH
```

```cpp
// ----------------------------------------------------------
// StdSimpleRun.hh
// ----------------------------------------------------------

//  The basic I/O unit is a run
//  The run contains methods to get the next event

#ifndef STDSIMPLERUN_HH
#define STDSIMPLERUN_HH

#include <vector>

#include "StdRun.hh"
#include "StdEvent.hh"

namespace StdHep  {

class SimpleRun: public Run {

public:

  // ---  birth/death:
  //
  inline SimpleRun( int Runno, int togen, int ngen, int nwrt,
                    double com, double xsec );
  inline SimpleRun( );
  inline SimpleRun( const SimpleRun & orig );
  inline SimpleRun& operator=( const SimpleRun & );
  ~SimpleRun() { ; }

  // ---  accessors:
  //
  int RunNumber()         const {  return itsRunNumber; }
  int EventsToGenerate() const {return itsEventsToGenerate; }
  int EventsGenerated()   const { return itsEventsGenerated; }
  int EventsWritten()     const { return itsEventsWritten; }
  double CoMEnergy()      const {  return itsCoMEnergy; }
  double CrossSection()   const {  return itsCrossSection; }
  double Seed( int i )    const  {  return itsSeedList[i]; }
  void PrintVersion();

  // ---  mutators:
  //
  void setRunNumber ( int runno )   { itsRunNumber = runno; }
  void setEventsToGenerate (int g) { itsEventsToGenerate=g; }
  void setEventsGenerated  (int n) { itsEventsGenerated=n; }
  void setEventsWritten ( int nw ) { itsEventsWritten = nw; }
  void setCoMEnergy ( double com ) { itsCoMEnergy = com; }
  void setCrossSection ( double x ) { itsCrossSection = x; }
  void appendSeed (double sd) { itsSeedList.push_back(sd); }
  void clearSeeds ( )     { itsSeedList.clear(); }
};       // SimpleRun
}        // StdHep
#endif // STDSIMPLERUN_HH
```