

# HepPDT

## 3.04.01

### Particle Data Table Classes

Lynn Garren, W. Brown, M. Fischler, M. Paterno

Sep. 30, 2009

[garren@fnal.gov](mailto:garren@fnal.gov)

<https://savannah.cern.ch/projects/heppdt/>

<http://cepa.fnal.gov/psm/heppdt/>

#### DISCLAIMER

The software is licensed on an "as is" basis only. Universities Research Association, Inc. (URA) makes no representations or warranties, express or implied. By way of example but not of limitation, URA makes no representations or WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE, that the use of the licensed software will not infringe any patent, copyright, or trademark, or as to the use (or the results of the use) of the licensed software or written material in terms of correctness, accuracy, reliability, currentness or otherwise. The entire risk as to the results and performance of the licensed software is assumed by the user. Universities Research Association, Inc. and any of its trustees, overseers, directors, officers, employees, agents or contractors shall not be liable under any claim, charge, or demand, whether in contract, tort, criminal law, or otherwise, for any and all loss, cost, charge, claim, demand, fee, expense, or damage of every nature and kind arising out of, connected with, resulting from or sustained as a result of using this software. In no event shall URA be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, fees or expenses of any nature or kind.

This material resulted from work developed under a Government Contract and is subject to the following license: The Government retains a paid-up, nonexclusive, irrevocable worldwide license to reproduce, prepare derivative works, perform publicly and display publicly by or for the Government, including the right to distribute to other Government contractors. Neither the United States nor the United States Department of Energy, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

For further information about this program, contact Lynn Garren, Fermi National Accelerator Laboratory (FNAL), (630) 840-2061, garren@fnal.gov. The HepPDT home page is <https://savannah.cern.ch/projects/heppdt/>.

# Contents

<b>1</b>	<b>Overview</b>	<b>3</b>
1.1	HepPDT Design . . . . .	3
1.2	HepPDT Classes . . . . .	3
<b>2</b>	<b>Particle Numbering Scheme</b>	<b>3</b>
2.1	ParticleID . . . . .	5
<b>3</b>	<b>Particle Properties</b>	<b>7</b>
3.1	Reading Particle Data information . . . . .	7
3.2	Accessing Particle Data information . . . . .	8
3.3	The Measurement Class . . . . .	9
3.4	Particles Not in the Table . . . . .	9
<b>4</b>	<b>Conclusions</b>	<b>9</b>
<b>A</b>	<b>ParticleID.hh</b>	<b>11</b>
<b>B</b>	<b>Input Data Examples</b>	<b>14</b>
B.1	Read PDG Particle Data . . . . .	14
B.2	Read particle.tbl . . . . .	15
B.3	Read EvtGen Particle Data . . . . .	16
B.4	Read Pythia Particle Data . . . . .	17
B.5	Read QQ Particle Data . . . . .	18
<b>C</b>	<b>Handling Unknown Particle ID's</b>	<b>20</b>
C.1	Abstract Base Class . . . . .	20
C.2	SimpleProcessUnknownID . . . . .	20
C.3	HeavyIonUnknownID . . . . .	21
C.4	Using MyProcessUnknownID . . . . .	22



# 1 Overview

For some time, there has been a need for a C++ class embodying the information contained in the Review of Particle Properties[1]. We have written HepPDT to fill this need. HepPDT allows access to particle name, particle ID, charge, nominal mass, total width, spin information, color information, and constituent particles. HepPDT is designed to be used by any Monte Carlo generated particle class. Generated particles could, if desired, contain a pointer to the particle data information found in the HepPDT particle data table.

## 1.1 HepPDT Design

HepPDT has been designed to be used by any Monte Carlo particle generator or decay package. It contains only generic particle attributes. In principle, all information which can be found in the Review of Particle Properties[1] can be encapsulated in HepPDT. However, because the user base has not found the inclusion of decay table information useful, HepPDT 3.xx no longer contains any decay information.

HepPDT stores particle information such as charge and nominal mass in a table which is accessed by a particle ID number. The particle ID number is defined according to the Particle Data Group's Monte Carlo numbering scheme[2].

Methods are provided to create ParticleDataTable objects from Pythia, Herwig, Isajet, QQ, and EvtGen. Methods are also provided to facilitate creation of custom particle information. A ParticleDataTable object may be created from multiple information sources.

The design requires that ParticleDataTable objects must be fully created before they are used. Multiple data tables are allowed. Although potentially dangerous, we recognize that this is also a powerful option.

Figure 1 shows the interactions of the basic classes.

## 1.2 HepPDT Classes

The ParticleDataTable class contains a map of ParticleData which is keyed on the ParticleID class.

The ParticleID class can be used to retrieve all the information that is implied in the particle ID (*e.g.*, charge and quark content). Boolean methods (such as isMeson, isBaryon, hasBottom, and hasTop) are provided for ease of searching for various types of particles.

The ParticleData class includes particle name, particle ID, charge, mass, total width with cutoffs, spin information, color information, and constituent particles (*e.g.*, quark content).

# 2 Particle Numbering Scheme

The Particle Data Group [1] provides a standard particle numbering scheme. This numbering scheme is described in full detail in reference [2].

HepPDT uses the translation methods in HepPID.[5]

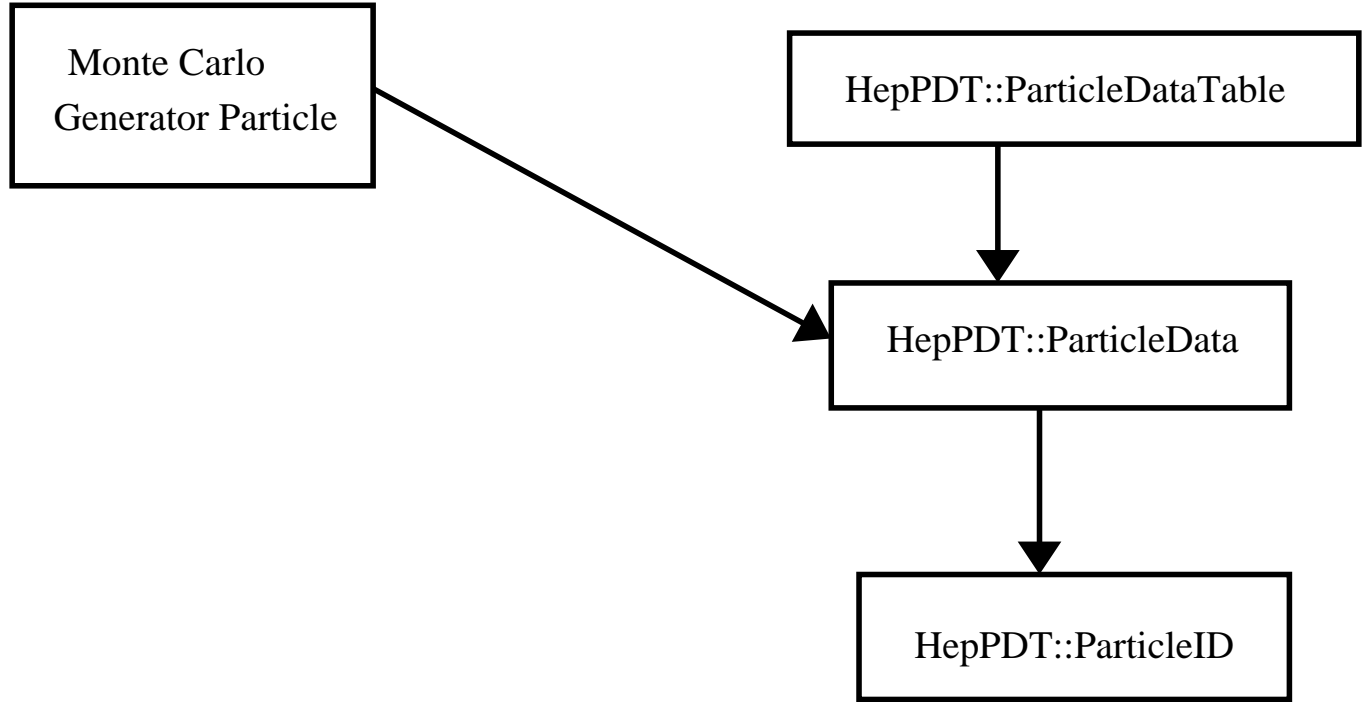


Figure 1: HepPDT Classes: Particle information is accessed either by a pointer to ParticleData from any Monte Carlo generated particle or by lookup with a string or ID. ParticleData contains particle information such as mass, charge, and total width. The ParticleDataTable contains a map of ParticleData objects, referenced by ParticleID. ParticleData has methods to access all relevant information.

## 2.1 ParticleID

The `HepPDT::ParticleID` class provides methods to return all the information which can be extracted or inferred from the particle ID (PID). It is expected that any 7 digit number used as a PID will adhere to the rules of the Monte Carlo Particle Numbering Scheme published by the PDG.[1]

The `ParticleID` class considers any particle with an ID less than 100 a "fundamental" particle.

In most cases, a user can define particles not already in the Particle Data Table without needing to extend the numbering scheme. A previously unknown particle can be assigned a valid PID by following the rules in the "Review of Particle Physics".[1]

If the user wishes to force the decay chain  $D^* \rightarrow D^0\pi^0$ ,  $D^0 \rightarrow K^-\pi^+\pi^0\pi^0$ , a user might define a special  $D^0$  which only decays to  $K^-\pi^+\pi^0\pi^0$ , leaving any  $D^0$  produced elsewhere to decay normally. The PID for a normal  $D^0$  is 421. The PID for the special  $D^0$  might be 6000421. This new PID might be used in several different jobs for  $D^0$  particles with different forced decay modes. (EvtGen defines these special particles as aliases in the decay table. The `TableBuilder` class handles the aliases appropriately without needing to create a new PID for the particle alias.)

	ParticleID( int pid = 0 );
	ParticleID( const ParticleID & orig );
	ParticleID & operator=( const ParticleID & );
void	swap( ParticleID & other );
bool	operator < ( ParticleID const & other ) const;
bool	operator == ( ParticleID const & other ) const;
int	pid( ) const;
int	abspid( ) const;
bool	isValid( ) const;
bool	isMeson( ) const;
bool	isBaryon( ) const;
bool	isDiQuark( ) const;
bool	isHadron( ) const;
bool	isLepton( ) const;
bool	isNucleus( ) const;
bool	isPentaquark( ) const;
bool	isSUSY( ) const;
bool	isRhadron( ) const;
bool	hasUp( ) const;
bool	hasDown( ) const;
bool	hasStrange( ) const;
bool	hasCharm( ) const;
bool	hasBottom( ) const;
bool	hasTop( ) const;
int	jSpin( ) const;
int	sSpin( ) const;
int	lSpin( ) const;
int	fundamentalID( ) const;
Quarks	quarks( ) const;
int	threeCharge( ) const;
double	charge( ) const;
int	A( ) const;
int	Z( ) const;
int	lambda( ) const;
unsigned short	digit(location) const;
const std::string	PDName() const;



## 3 Particle Properties

Particle data information is stored in `HepPDT::ParticleDataTable`, which is a map of `HepPDT::ParticleData` objects that are referenced by `HepPDT::ParticleID`. The design envisions that generated particles will contain links to the relevant `HepPDT::ParticleData` object.

### 3.1 Reading Particle Data information

HepPDT can accept particle data information from a variety of sources. To fill `HepPDT::ParticleDataTable`, the user creates an empty `ParticleDataTable` object and then calls `HepPDT::TableBuilder` methods to read the information from an input stream. Information may be read from as many input streams as desired. In case of conflicts, previous information will be overwritten. All information is kept in temporary objects until the `TableBuilder` destructor is called. The `TableBuilder` destructor then creates the `ParticleData` objects owned by `ParticleDataTable`.

The following code fragment reads Pythia input from a flat file. Examples reading input from other sources are in Appendix B and also in the example subdirectory.

```
#include <fstream>

#include "HepPDT/TableBuilder.hh"
#include "HepPDT/ParticleDataTable.hh"
#include "HepPDT/TempParticleData.hh"

    const char infile[] = "data/pythia.tbl";
    // open input file
    std::ifstream pdfile( infile );
    if( !pdfile ) {
        std::cerr << "cannot open " << infile << std::endl;
        exit(-1);
    }
    // construct empty PDT
    HepPDT::ParticleDataTable datacol( "Pythia Table" );
    {
        // Construct table builder
        HepPDT::TableBuilder  tb(datacol);
// read the input - put as many here as you want
        if( !addPythiaParticles( pdfile, tb ) )
            { std::cout << "error reading pythia file " << std::endl; }
    } // the tb destructor fills datacol
```

The Particle Data Group provides a table of particle masses and widths for known particles. This table, `pdg_mass.tbl`, is distributed with the HepPDT package. This information is also available from specific generators, often as flat files.

By request, a simple table, `particle.tbl`, of particle masses and widths has been added to HepPDT. This table is intended to be a complete list of useful particles. Use the `addParticleTable()` free function, defined in `TableBuilder.hh`, to parse this file.

## 3.2 Accessing Particle Data information

The following code fragment accesses pion and muon information. Refer to the Appendices for a listing of particle ID numbers.

```
std::ofstream wpdf( outfile );
HepPDT::ParticleDataTable db( "my Table" );
.....
HepPDT::ParticleData * pd = datacol.particle( HepPDT::ParticleID(111) );
pd->write(wpdf);
double mumass = datacol.particle( HepPDT::ParticleID(13) )->mass();
```

In principle, all information in the PDG may be obtained from ParticleData access methods.

std::string const &	name() const;
ParticleID	ID() const;
int	pid( ) const;
double	charge() const;
double	color() const;
SpinState	spin() const;
Measurement	mass() const;
Measurement	totalWidth() const;
Measurement	lifetime() const;
int	numConstituents() const;
Constituent	constituent( unsigned int i ) const;
ParticleID	constituentParticle( unsigned int i ) const;
ResonanceStructure const *	resonance() const;
bool	isMeson( ) const;
bool	isBaryon( ) const;
bool	isDiQuark( ) const;
bool	isHadron( ) const;
bool	isLepton( ) const;
bool	isNucleus( ) const;
bool	isPentaquark( ) const;
bool	isSUSY( ) const;
bool	isRhadron( ) const;
bool	isDyon( ) const;
bool	isQBall( ) const;
bool	hasUp() const;
bool	hasDown() const;
bool	hasStrange() const;
bool	hasCharm() const;
bool	hasBottom() const;
bool	hasTop() const;
void	write( std::ostream & os ) const;

### 3.3 The Measurement Class

Some tables contain errors on mass and width values. To keep this error information available, we wrote a simple `HepPDT::Measurement` class which contains a double value and a double error on the value. If you reference it with a double, `Measurement` returns the value.

```
Measurement( double value, double sigma );
Measurement( const Measurement & orig );
Measurement & operator=( const Measurement & );
void        swap( Measurement & other );
bool        operator < ( Measurement const & other ) const;
bool        operator == ( Measurement const & other ) const;
double      value() const;
double      sigma() const;
operator double() const;
```

### 3.4 Particles Not in the Table

If a particle definition has not been added to the `ParticleDataTable`, a lookup of that particle with either `operator[]` or the `particle()` method will return a null pointer to `ParticleData`. However, there are some instances where you might want to create a `ParticleData` dynamically. Specifically, this is handy when dealing with the heavy ions produced dynamically by `Geant4`.

An abstract plugin class, `ProcessUnknownID`, is now available that allows you to specify what happens when you try to lookup a particle ID that is not in the table. The plugin class contains `processUnknownID()` which returns a pointer to a `ParticleData` object. This pointer should be null if no `ParticleData` object is defined. The unknown `ParticleID` and a const reference to `ParticleDataTable` are passed to `processUnknownID()`. To use this class, create your own `MyProcessUnknownID` class which inherits from `ProcessUnknownID`. You only need to define a constructor and `MyProcessUnknownID::processUnknownID( ParticleID, const ParticleDataTable & )`.

By default, `ParticleDataTable` behaves exactly as before, returning a null pointer to `ParticleData`. `HepPDT` also contains the `HeavyIonUnknownID` class which will create a `ParticleData` for an unknown nuclear fragment. Code fragments are shown in appendix C.

## 4 Conclusions

`HepPDT` provides access to all useful particle data properties and is designed to be used with any generated particle. The `HepPDT` home page is <http://lcgapp.cern.ch/project/simu/HepPDT/>.

## References

- [1] <http://pdg.lbl.gov/>
- [2] Particle Data Group: C. Amsler *et al.*, *Physics Letters* **B667**, (2008) 1,  
[http://pdg.lbl.gov/2009/mcdata/mc\\_particle\\_id\\_contents.html](http://pdg.lbl.gov/2009/mcdata/mc_particle_id_contents.html)

- [3] Particle Data Group: W.-M. Yao *et al.*, *J. Phys.* **G 33**, 314 (2006),  
[http://pdg.lbl.gov/2006/mcdata/mc\\_particle\\_id\\_contents.html](http://pdg.lbl.gov/2006/mcdata/mc_particle_id_contents.html)
- [4] Particle Data Group: S. Eidelman *et al.*, *Physics Letters* **B592**, (2004) 292,  
[http://pdg.lbl.gov/2004/mcdata/mc\\_particle\\_id\\_contents.html](http://pdg.lbl.gov/2004/mcdata/mc_particle_id_contents.html)
- [5] <http://cepa.fnal.gov/psm/HepPID/>

# A ParticleID.hh

namespace HepPDT

Free functions:

```
double spinitod( int js );  
int spindtoi( double spin );
```

Public members:

```
enum location { nj=1, nq3, nq2, nq1, nl, nr, n, n8, n9, n10 };  
struct Quarks {  
    Quarks( ) : nq1(0), nq2(0), nq3(0) {}  
    Quarks( short q1, short q2, short q3 ) : nq1(q1), nq2(q2), nq3(q3) {}  
    short nq1; short nq2; short nq3; };
```

CLASS ParticleID

Public Methods:

```
ParticleID( int pid = 0 );  
    The constructor.  
ParticleID( const ParticleID & orig );  
    The copy constructor.  
ParticleID & operator=( const ParticleID & );  
    The assignment constructor.  
void swap( ParticleID & other );  
    The swap constructor.  
bool operator < ( ParticleID const & other ) const;  
    Comparison operator.  
bool operator == ( ParticleID const & other ) const;  
    Equality operator.  
int pid( ) const;  
    Returns the PID.  
int abspid( ) const;  
    Returns the absolute value of the PID.  
bool isValid( ) const;  
    Returns true if this integer obeys the numbering scheme rules.  
bool isMeson( ) const;  
    Returns true if this integer obeys the meson portion of the numbering scheme rules.  
bool isBaryon( ) const;  
    Returns true if this integer obeys the baryon portion of the numbering scheme rules.  
bool isDiQuark( ) const;  
    Returns true if this integer obeys the diquark portion of the numbering scheme rules.  
bool isHadron( ) const;  
    Returns true if either isBaryon or isMeson is true.  
bool isLepton( ) const;
```

Returns true if the fundamentalID is 11-18.

**bool isNucleus( ) const;**  
Returns true if this integer obeys the ion numbering scheme rules.

**bool isPentaquark( ) const;**  
Returns true if this integer obeys the pentaquark numbering scheme rules.

**bool isSUSY( ) const;**  
Returns true if this integer obeys the SUSY numbering scheme rules.

**bool isRhadron( ) const;**  
Returns true if this integer obeys the R-hadron numbering scheme rules.

**bool isDyon( ) const;**  
Returns true if this integer obeys the dyon numbering scheme rules.

**bool isQBall( ) const;**  
Returns true if this integer obeys the ad-hoc Q-ball numbering scheme rules.

**bool hasUp( ) const;**  
Returns true if this is a valid PID and it has an up quark.

**bool hasDown( ) const;**  
Returns true if this is a valid PID and it has a down quark.

**bool hasStrange( ) const;**  
Returns true if this is a valid PID and it has a strange quark.

**bool hasCharm( ) const;**  
Returns true if this is a valid PID and it has a charm quark.

**bool hasBottom( ) const;**  
Returns true if this is a valid PID and it has a bottom quark.

**bool hasTop( ) const;**  
Returns true if this is a valid PID and it has a top quark.

**int jSpin( ) const;**  
jSpin returns  $2J+1$ , where J is the total spin

**int sSpin( ) const;**  
sSpin returns  $2S+1$ , where S is the spin

**int lSpin( ) const;**  
lSpin returns  $2L+1$ , where L is the orbital angular momentum

**int fundamentalID( ) const;**  
Returns the first 2 digits if this is a valid PID and it is neither neither a meson, a baryon, nor a diquark. If this is a meson, baryon, or diquark, fundamentalID returns zero.

**int extraBits( ) const;**  
Returns any digits beyond the 7th digit (e.g. outside the numbering scheme).

**Quarks quarks( ) const;**  
Returns a struct with the 3 quarks.

**int threeCharge( ) const;**  
Returns 3 times the charge, as inferred from the quark content.

**double charge( ) const;**  
Returns the actual charge, as inferred from the quark content.  
If the fundamentalID is non-zero, then a lookup table is used.

**int A( ) const;**  
If this is an ion, returns A.

**int Z( ) const;**

    If this is an ion, returns Z.

**int lambda( ) const;**

    If this is an ion, returns nLambda.

**unsigned short digit(location) const;**

    digit returns the base 10 digit at a named location in the PID

**const std::string PDTname() const;**

    Returns the HepPDT standard name.

**Private Members:**

**int itsPID;**

## B Input Data Examples

### B.1 Read PDG Particle Data

```
// -----  
// testHepPDT.cc  
// Author: Lynn Garren  
//  
// read PDG table and write it out  
//  
// -----  
  
#include <fstream>  
  
#include "HepPDT/defs.h"  
#include "HepPDT/TableBuilder.hh"  
#include "HepPDT/ParticleDataTable.hh"  
  
int main()  
{  
    const char pdgfile[] = "../data/mass_width_2006.mc";  
    const char outfile[] = "PDfile";  
    // open input file  
    std::ifstream pdfile( pdgfile );  
    if( !pdfile ) {  
        std::cerr << "cannot open " << pdgfile << std::endl;  
        exit(-1);  
    }  
    // construct empty PDT  
    HepPDT::ParticleDataTable datacol( "PDG Table" );  
    {  
        // Construct table builder  
        HepPDT::TableBuilder tb(datacol);  
        // read the input - put as many here as you want  
        if( !addPDGParticles( pdfile, tb ) )  
        { std::cout << "error reading PDG file " << std::endl; }  
        } // the tb destructor fills datacol  
        std::ofstream wpdfile( outfile );  
        if( !wpdfile ) {  
            std::cerr << "cannot open " << outfile << std::endl;  
            exit(-1);  
        }  
        datacol.writeParticleData(wpdfile);  
        wpdfile << std::endl;  
    }
```



```

    return 0;
}

```

## B.2 Read particle.tbl

```

// -----
// testReadParticleTable.cc
//
// read particle.tbl and write it out
//
// -----

#include <fstream>

#include "HepPDT/defs.h"
#include "HepPDT/TableBuilder.hh"
#include "HepPDT/ParticleDataTable.hh"

int main()
{
    const char infile[] = "../data/particle.tbl";
    const char outfile[] = "testReadParticleTable.out";
    // open input files
    std::ifstream pfile( infile );
    if( !pfile ) {
        std::cerr << "cannot open " << infile << std::endl;
        exit(-1);
    }
    // construct empty PDT
    HepPDT::ParticleDataTable datacol( "Generic Particle Table" );
    {
        // Construct table builder
        HepPDT::TableBuilder tb(datacol);
// read the input - put as many here as you want
        if( !addParticleTable( pfile, tb ) ) { std::cout << "error reading EvtGen pdt f
    } // the tb destructor fills datacol
    // open the output stream
    std::ofstream wfile( outfile );
    if( !wfile ) {
        std::cerr << "cannot open " << outfile << std::endl;
        exit(-1);
    }
    // write the data table
    datacol.writeParticleData(wfile);
    return 0;
}

```

```
}
```

### B.3 Read EvtGen Particle Data

```
// -----  
// testReadEvtGen.cc  
//  
// read EvtGen table and write it out  
//  
// -----  
  
#include <fstream>  
  
#include "HepPDT/defs.h"  
#include "HepPDT/TableBuilder.hh"  
#include "HepPDT/ParticleDataTable.hh"  
  
int main()  
{  
    const char infile1[] = "../examples/data/evt.pdl";  
    const char infile2[] = "../examples/data/DECAY.DEC";  
    const char outfile[] = "testReadEvtGen.out";  
    // open input files  
    std::ifstream pdfile1( infile1 );  
    if( !pdfile1 ) {  
        std::cerr << "cannot open " << infile1 << std::endl;  
        exit(-1);  
    }  
    // construct empty PDT  
    std::ifstream pdfile2( infile2 );  
    if( !pdfile2 ) {  
        std::cerr << "cannot open " << infile2 << std::endl;  
        exit(-1);  
    }  
    HepPDT::ParticleDataTable datacol( "EvtGen Table" );  
    {  
        // Construct table builder  
        HepPDT::TableBuilder tb(datacol);  
    // read the input - put as many here as you want  
        if( !addEvtGenParticles( pdfile1, tb ) ) { std::cout << "error reading EvtGen pd  
        if( !addEvtGenParticles( pdfile2, tb ) ) { std::cout << "error reading EvtGen de  
    } // the tb destructor fills datacol  
    std::ofstream wfile( outfile );  
    if( !wfile ) {  
        std::cerr << "cannot open " << outfile << std::endl;  
    }  
}
```

```

        exit(-1);
    }
    datacol.writeParticleData(wfile);

    return 0;
}

```

## B.4 Read Pythia Particle Data

```

// -----
// listPythiaNames.cc
// Author: Lynn Garren
//
// read Pythia table and write it out
//
// -----

#include <fstream>
#include <iostream>

#include "HepPDT/TableBuilder.hh"
#include "HepPDT/ParticleDataTable.hh"

int main()
{
    const char infile[] = "../listPythia.tbl";
    const char outfile[] = "listPythiaNames.out";
    // open input file
    std::ifstream pdfile( infile );
    if( !pdfile ) {
        std::cerr << "cannot open " << infile << std::endl;
        exit(-1);
    }
    // construct empty PDT
    HepPDT::ParticleDataTable datacol( "Pythia Table" );
    {
        // Construct table builder
        HepPDT::TableBuilder tb(datacol);
        // read the input - put as many here as you want
        if( !addPythiaParticles( pdfile, tb ) )
        { std::cout << "error reading pythia file " << std::endl; }
        // the tb destructor fills datacol
        // open output file
        std::ofstream wpdfile( outfile );
        if( !wpdfile ) {

```

```

        std::cerr << "cannot open " << outfile << std::endl;
        exit(-1);
    }
    // write the particle and decay info
    datacol.writeParticleData( wpdfilename );

    return 0;
}

```

## B.5 Read QQ Particle Data

```

// -----
// testReadQQ.cc
//
// read QQ table and write it out
//
// -----

#include <fstream>

#include "HepPDT/defs.h"
#include "HepPDT/TableBuilder.hh"
#include "HepPDT/ParticleDataTable.hh"

int main()
{
    const char infile[] = "../listQQ.dec";
    const char outfile[] = "testReadQQ.out";
    // open input file
    std::ifstream pdfilename( infile );
    if( !pdfilename ) {
        std::cerr << "cannot open " << infile << std::endl;
        exit(-1);
    }
    // construct empty PDT
    HepPDT::ParticleDataTable datacol( "QQ Table" );
    {
        // Construct table builder
        HepPDT::TableBuilder tb(datacol);
        // read the input - put as many here as you want
        if( !addQQParticles( pdfilename, tb ) )
            { std::cout << "error reading QQ table file " << std::endl; }
        // the tb destructor fills the PDT
        std::ofstream wpdfilename( outfile );
        if( !wpdfilename ) {

```

```
        std::cerr << "cannot open " << outfile << std::endl;
        exit(-1);
    }
    // write the particle and decay info
    datacol.writeParticleData( wpdfilename );

    return 0;
}
```

## C Handling Unknown Particle ID's

### C.1 Abstract Base Class

The arguments for `processUnknownID()` must be set so that any inheriting class has all necessary information. We pass a const reference to the `ParticleDataTable` instance.

```
namespace HepPDT {

// forward declaration to avoid circular dependencies
class ParticleDataTable;

class ProcessUnknownID {

public:

    /// safety wrapper to avoid secondary calls to processUnknownID
    ParticleData * callProcessUnknownID( ParticleID, const ParticleDataTable & );

    /// allow cleanup by ParticleDataTable
    virtual ~ProcessUnknownID( ) {}

protected:
    ProcessUnknownID( ) : alreadyHere(false) {}

private:

    bool    alreadyHere;
    virtual ParticleData * processUnknownID( ParticleID,
                                              const ParticleDataTable & ) = 0;

}; // ProcessUnknownID

} // HepPDT
```

### C.2 SimpleProcessUnknownID

This is the default implementation of `processUnknownID()`. Notice that it simply returns a null pointer.

```
namespace HepPDT {

class SimpleProcessUnknownID : public ProcessUnknownID {
public:
    SimpleProcessUnknownID() {}
```

```

    virtual ParticleData * processUnknownID
        ( ParticleID key, const ParticleDataTable & pdt )
    { return 0; }

};

}          // HepPDT

```

### C.3 HeavyIonUnknownID

HeavyIonUnknownID creates and returns a pointer to a ParticleData. This processUnknownID method only uses the proton mass to calculate an approximate mass for the nuclear fragment.

```

// HeavyIonUnknownID.hh
namespace HepPDT {

class HeavyIonUnknownID : public ProcessUnknownID {
public:
    HeavyIonUnknownID() {}

    virtual ParticleData * processUnknownID( ParticleID, const ParticleDataTable & pdt )

};

}          // HepPDT

// HeavyIonUnknownID.cc
namespace HepPDT {

ParticleData * HeavyIonUnknownID::processUnknownID
    ( ParticleID key, const ParticleDataTable & pdt )
{
    if( ! key.isNucleus() ) return 0;

    // have to create a TempParticleData with all properties first
    TempParticleData tpd(key);
    // calculate approximate mass
    // WARNING: any calls to particle() from here MUST reference
    //           a ParticleData which is already in the table
    // This convention is enforced.
    const ParticleData * proton = pdt.particle(2212);
    if( ! proton ) return 0;
    double protonMass = proton->mass();
    tpd.tempMass = Measurement(key.A()*protonMass, 0.);

```

```

        return new ParticleData(tpd);
    }

    // HepPDT

```

## C.4 Using MyProcessUnknownID

To use HeavyIonUnknownID, you simply need to specify it when you create your ParticleDataTable object:

```

#include "HepPDT/HeavyIonUnknownID.hh"
...
HepPDT::ParticleDataTable pdt( "Handle Heavy Ions",
                               new HepPDT::HeavyIonUnknownID );

```

You may also create your own method and call it in the same way.