EvtGen
A Monte Carlo generator for B-physics

Anders Ryd (ryd@hep.caltech.edu)
David Lange (lange@charm.physics.ucsb.edu)
Natalia Kuznetsova (natalia@charm.physics.ucsb.edu)
Sophie Versille (versille@in2p3.fr)
Marcello Rotondo (Marcello.Rotondo@roma1.infn.it)
Alain Bellerive (alainb@lns101.lns.cornell.edu)
Veronique Boisvert (vb20@lns598.lns.cornell.edu)
November 6, 1998

Abstract

This note describes EvtGen, an event generator that is well suited for B physics. It implements many detailed models that are important for the physics of B mesons. In particular, it has detailed models for semileptonic decays, CP violating decays and produces correct results for the angular distributions in sequential decays, including all correlations. It also has an interface to JetSet for generation of continuum at the $\Upsilon(4S)$ and for generic hadronic decays, e.g. of Bs, that are not implemented in the generator.

Contents

1 Introduction

There are several event generators available for the simulation of particle decays in high energy physics experiments. Examples used for the simulation of B physics include the well known packages QQ [?] and JETSET [?]. This paper describes a new package that we hope will be a useful tool for the simulation of decays of B mesons and other resonances. With several new B physics experiments planning to take data in the next few years, the physics of B mesons will be studied in greater detail than previously possible. It is important to have tools for the simulation of the underlying physics processes at these experiments.

The EvtGen package provides a framework in which new decays can be added as modules. These modules, which perform the simulation of decays, are called models in EvtGen. One of the novel ideas in the design of EvtGen is that decay amplitudes, instead of probabilities, are used for the simulation of complex decays. The framework uses the amplitude for each node in the decay tree to simulate the entire decay chain, including all angular correlations. A few examples of sequential decays are

$$
\begin{array}{lll}
B \to D^*\ell\nu & \qquad & B \to D^*\ell\nu \\
\qquad \hookrightarrow D\pi & & \qquad \hookrightarrow D\gamma \\
 & & \\
B \to D^* \quad D^* & & B \to D^* \quad D^* \\
\qquad \hookrightarrow D\pi \hookrightarrow D\pi & & \qquad \hookrightarrow D\pi \hookrightarrow D\gamma
\end{array}
$$

For these decays, the EvtGen framework models all decay distributions correctly, while implementing only the nodes ($B \to D^*\ell\nu$, $D^* \to D\pi$, etc.) of the decay trees. CP violating decays have their own particular challenges, including non-trivial decay time distributions. Examples of distributions from EvtGen for the decay $B \to J/\psi K^*$ are shown in Figure 1.

This document will describe the functionality and organization of the EvtGen package. Additionally, we will discuss how to use EvtGen, as well as how to implement new physics models. EvtGen is written in C++,contains about 125 classes and 25000 lines of code. There are approximately 60 models implemented that simulate a large variety of physics processes. The modularity of the code allows for the easy implementation of additional models.

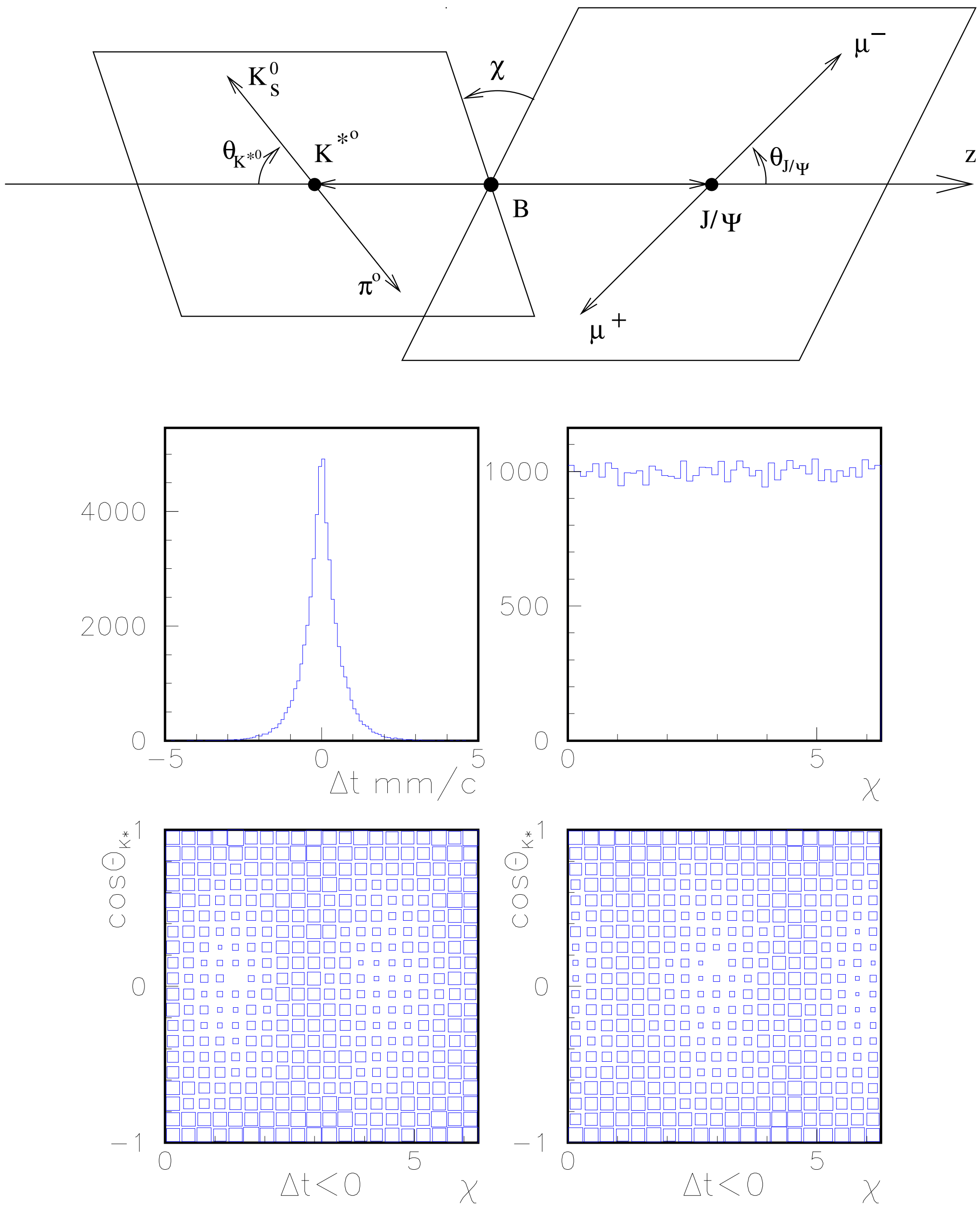

Figure 1: The top diagram defines the angles in the decay $B \to J/\psi K^*$ with $J/\psi \to \mu^+\mu^-$ and $K^{*0} \to K_S^0 \pi^0$. The lower four plots shows projections of the distributions from an EvtGen simulation of this decay. In this simulation the B meson was produced in an $\Upsilon(4S)$ decay and Δt is the difference in proper lifetimes of the two B mesons. The lower four plots shows (clockwise from the upper left) the time asymmetry, the χ angle distribution, χ vs. $\cos\theta_{K^*}$ for $\Delta t < 0$, and χ vs. $\cos\theta_{K^*}$ for $\Delta t > 0$.

2 EvtGen.tar

There are a handfull of files other than the .cc files included in the `EvtGen.tar` file. The functionality of the `pdt.table`, `DECAY.DEC`, `configure`, and `Makefile` files is described below. Additionally, a `test` directory is included which provides several diagnostic tests of the EvtGen library.

The `pdt.table` is needed at runtime by the EvtGen package. It contains the list of particles and particle properties to be used. This table is fully described in Section 4. Similarly, the `DECAY.DEC` contains the default list of particle decay channels, including branching ratios, to be used by EvtGen. `DECAY.DEC` is described in Section 5.

`Makefile` is responsible for building the EvtGen library as well as test executables. The unpacking of EvtGen.tar and the use of the `Makefile` are described in Section 3. `configure` is responsible for properly setting up the paths for the `Makefile`.

The directory `test` contains the files necesary to perform several tests of the EvtGen code. Performing these tests is also detailed in Section 3

3 Get started with EvtGen

After unpacking the `EvtGen.tar` file, it is quite simple to get started with EvtGen. This section will describe the process of compiling and linking code for several simple tests. Sample output of these tests is shown. Manipulating the output of EvtGen using user decay files is also explained.

To properly set up the `Makefile`, type `configure`. The `configure` script will ask several questions, and then build the `Makefile`. The output of the configure script will look something like the following:

```
neutrino> configure
Welcome to configure for EvtGen
This machine is running: OSF1
OSF1 is supported.
Enter Name of c++ compiler (ie c++,cxx,CC, etc)
cxx
Will use cxx for compilation of C++ code
Enter name of fortran comiler
f77
Will use f77 for compilation of fortran code
The EvtGen package uses photos, jetset74, and hbook from cernlib
Enter path to cernlib (e.g. /cern/pro/
/cern/pro/
You entered /cern/pro/ as path for cernlib
Found /cern/pro//lib/libjetset74.a
Found /cern/pro//lib/libphotos.a
Found /cern/pro//lib/libpacklib.a
Enter path to CLHEP
/nfs/babar2/ryd/pkg/
You entered /nfs/babar2/ryd/pkg/ as path
Found /nfs/babar2/ryd/pkg//include
Found /nfs/babar2/ryd/pkg//lib/libCLHEP.a
Finding compiler includes
The cxx compiler includes are
-I/usr/include/cxx -I/nfs/babar2/ryd/pkg//include
The default cxx options are
-I.. -I.
Enter any other options that you would like
to use in compilation.  Examples -g -pg -o4 etc

The following options will be used by default
-I.. -I.
The default link options are
for the test program.
Please enter any others that you would like to use

neutrino>
```

To build the EvtGen libary simply type `gmake` in the `EvtGen` directory. This should compile each `.cc` file and make a library called `libEvtGen.a`. If this fails to compile on your platform, please contact the authors for help. To compile EvtGen with no changes, the JETSET [?] and PHOTOS [?] packages from CERNLIB are needed, as well as the CLHEP [?] package.

If the library was successfully built, type `gmake test1` in order to built a sample executable (called `testEvtGen`). With this executable, several tests of the EvtGen library may be performed to check that the code is working properly. To do these type

```
cd test
../testEvtGen test1 1000
../testEvtGen ddalitz 1000
```

the `test` directory should now contain `test1.hst` and `ddalitz.hst`, each of which are HBOOK files containing several histograms. Figures 2 and 3 show some of these histograms.

Alternatively, to compile a sample executable using the fortran interface, type `gmake test2`. An executable called `fortranEvtGen` will be created. The fortran interface routine is contained in `myfort.F`

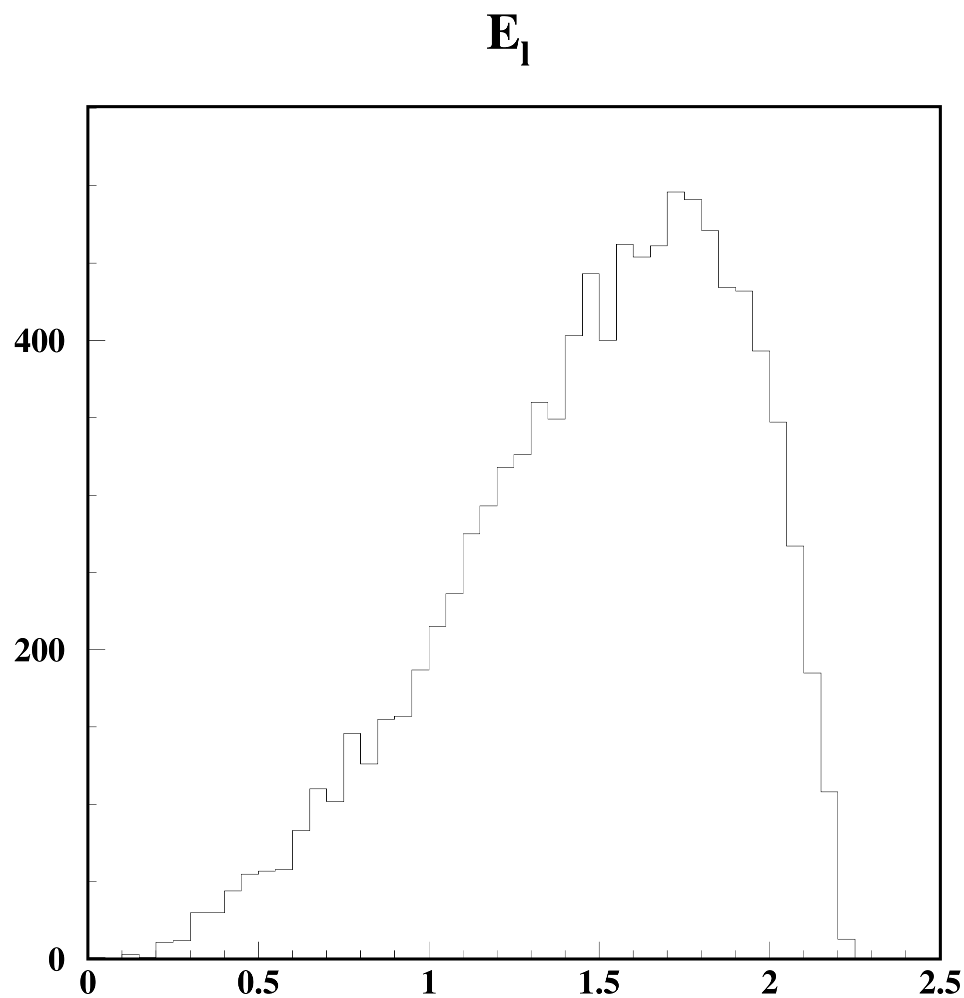

Figure 2: Histogram number 3 in `test1.hst` for 10000 events

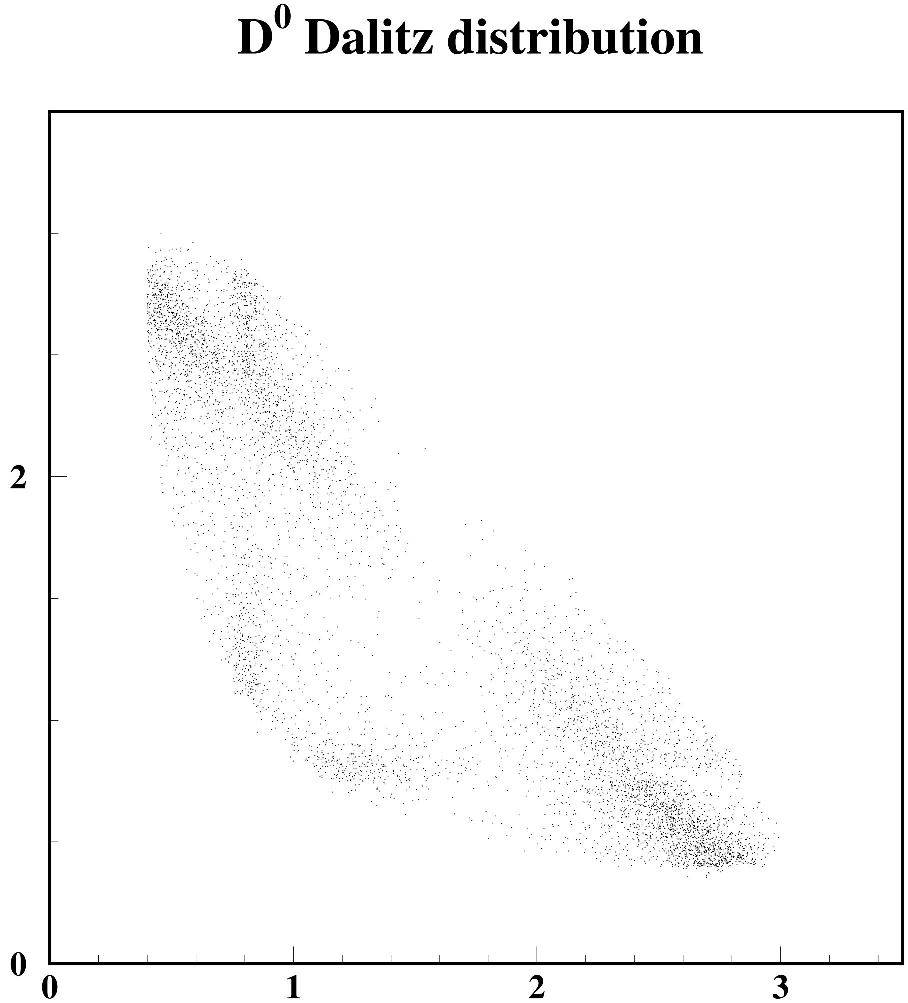

Figure 3: Histogram number 1 in ddalitz.hst for 10000 events

4 Particle properties in EvtGen

In EvtGen, all particle property information is contained within `pdt.table` and is parsed at runtime. Each line in `pdt.table` corresponds to a particle, and has the form

```
add  p Lepton  mu-       13  0.1056584  0     0   -3  1 658654.  13
add  p Lepton  mu+      -13  0.1056584  0     0    3  1 658654.   0
add  p Meson   pi+      211  0.139570   0     0    3  0  7804.5 101
add  p Meson   pi-     -211  0.139570   0     0   -3  0  7804.5   0
add  p Meson   rho+     213  0.7685     0.151 0.4  3  2       0 121
add  p Meson   rho-    -213  0.7685     0.151 0.4 -3  2       0   0
add  p Meson   D0       421  1.86451    0     0    0  0  0.1244 105
add  p Meson   anti-D0 -421  1.86451    0     0    0  0  0.1244   0
```

The first three columns are not used in EvtGen. The fourth column corresponds to the particle name. The fifth column is the particle number. Here we have used StdHep numbers. The sixth through eighth columns contain the mass, width, and maximum allowed deviation from the nominal mass, respectively. The nineth column contains 3 times the charge of the particle. The tenth column contains twice the spin. The eleventh column is $c\tau$ in meters. The twelvth column is the Lund-KC number *huh????*.

5 Decay tables in EvtGen

The decay table included in the EvtGen package, `DECAY.DEC`, provides a reasonably complete decay table for resonances below the $\Upsilon(4S)$. To describe the format of the decay table consider the decay of a D^{*+} meson

```
Decay D*+
0.67  D0 pi+     VSS;
0.33  D+ pi0     VSS;
Enddecay
```

A decay entry starts with the keyword "Decay" and ends with "Enddecay". Throughout the decay table, capitalization is important. Between these keywords, each line in the decay table corresponds to a decay channel. The first piece of information is the branching ratio. Following the branching ratio, the daughters are listed. For many models, the order of the daughters is important. The correct order for each model can be found in the `DECAY.DEC` provided, or in Section A of this documentation. Finally, the model for each decay channel is given. Again, details on available models can be found in Section A. The entry for each decay channel must end in a semicolon (";").

Certain models also take arguments, as in the case of the decay $B^0 \to \pi^+\pi^-$

```
Decay B0
1.00  pi+ pi-   SSS_CP dm alpha 1 1.0 0.0 1.0 0.0;
Enddecay
```

The meaning of the arguments for this model are explained in Section A.

The above example also shows the use of constants defined in the decay table, in this case "dm" and "alpha". The syntax for defining constants is

```
Define alpha 0.9
```

Note that a "Define" statement can not be within a "Decay-Enddecay" statement. We have followed the convention that all constants should be defined at the top of the decay table, however this is not necesary. If a constant is redefined, the last value defined is used.

Another feature implemented is that particles may be "Alias"-ed. This feature is useful for various reasons, for example when generating Monte Carlo for special signal channels. Consider the case of generating Monte Carlo for the decay $B^0 \to J/\Psi K^0_s$ with $J/\Psi \to \mu^+\mu^-$. Not all J/Ψs in the event should decay to $\mu^+\mu^-$, only the ones in the signal mode. The concept of particle aliases solves this problem. Consider the example of

```
Alias myJ/psi  Jpsi
Decay B0
1.00   myJ/psi  K_S0     SVS_CP dm beta 1.0 1.0 0.0 1.0 0.0;
Endedecay
Decay myJ/psi
1.00   mu+   mu-         VLL;
Enddecay
```

Here the particle myJ/psi has been aliased to the J/psi. Inside the generator, myJ/psi is treated as a new particle, with the same properties as the J/psi, except for the decay table. This includes the particle name and number, such that at the end of each event, the output will contain the chain $B^0 \to J/\Psi K^0_s$, $J/\Psi \to \mu^+\mu^-$. There is no need for analysis code to know about myJ/psi.

Once the decays of a particle have been specified, say the D^0, the charge conjugate decay modes can be specified for the anti-particle ($\bar{D}^0$) using the syntax

```
CDecay anti-D0.
```

Final state radiation using the PHOTOS [?] package may be included in each decay chain. This option is invoked by the key word PHOTOS which is placed after the list of decay daughters, but before the model name. See Section C for more details.

The keyword JetSetPar allows for manipulation of the parameters in the JETSET common blocks using the lugive subroutine. To set the paremeter MSTJ(26)=0 (this turns off B^0-$\bar{B}^0$ mixing) the line

```
JetSetPar MSTJ(26)=0
```

should be added to the decay table. Note that no spaces are allowed in the string that is passed to lugive. This is due to a limitation of how the decay table is parsed by EvtGen.

There are several things to know when writing a decay table:

- The decay table must end with End. This is also true for user decay tables as described in Section 3.
- Everything in the decay table is case sensitive.
- Charge conservation will be checked on each decay in the decay table.
- Each parent and daughter particle must be contained in pdt.table.
- Most models included in the EvtGen package will check that the number of daughters as well as the spin of each daughter are correct.
- The number of arguements provided must be the same as is expected by the model.
- If the sum of branching ratios in the decay channels for a particle do not sum to 1.0, they will all be rescaled so that the sum is 1.0.
- Any line beginning with "#" is considered a comment.

In addition to changing DECAY.DEC, the output of EvtGen may be controlled via a user decay file. We now give several examples of how to configure a user decay table. Many details of decay tables have already been discussed above, in Section 5.

As the first example, consider the example of generating $B^+ \to D^{*0} e^+ \nu_e$ with $D^{*0} \to \bar{D}^0 \pi^0$ and $\bar{D}^0 \to K^+\pi^-$. The other B from the $\Upsilon(4S)$ decays generically.

The standard way of using decay tables begins with parsing the full standard decay table, DECAY.DEC. After this, a user decay table is parsed in order to redefine the decays of particles of interest. This way, the user decay file will override the standard decay table.

In the example above, the user decay table could be implemented as:

```
#  Ups(4S)->B+ B-
#          |   |-> generic
#          |->D*0 e+ nu_e
#             |->D0B pi0
#                |->K+pi-
#
Alias myD*0      D*0
Alias my-anti-D0 anti-D0
#
Decay  Upsilon(4S)
1.000  B+      B-                  VSS;
Enddecay
#
Decay  B+
1.000  my-anti-D*0   e+   nu_e            ISGW2;
Enddecay
#
Decay  my-anti-D*0
1.000  my-anti-D0 pi0                 VSS;
Enddecay
#
Decay  my-anti-D0
1.000  K+      pi-                 PHSP;
Enddecay
#
End
```

The decay of the $\Upsilon(4S)$ is redefined. In the default decay table it is defined to decay to a proper mixture of charged and neutral Bs. Note that when a `Decay` statement is found for a particle, it erases all previous decays that have been defined for that particle. The $\Upsilon(4S)$ above is redefined such that it decays into a B^0 and a $\bar{B}^0$ 100% of the time. The $\bar{B}^0$ decay is not redefined and hence it decays generically according to the entries in DECAY.DEC. However, the B^0 is redefined such that it is forced to decay semileptonically according to the model of ISGW2. (For more information about details about what the different models do see Section A.)

Additionally, two new particles are created, the `myD*0` and the `my-anti-D*0`. As described in Section 5, the generator will treat these as new particles, which have the same properties as the original particle, except for the decay table. This means that the masses, widths, lifetime, etc. are the same but the list of what they decay to is different. After the generator is done with the event, the aliases are replaced with the original particle identifiers, such that further code need not know about the aliased particles. It is not necesary to alais either the B^+ or $\Upsilon(4S)$, as all B^+ and $\Upsilon(4S)$ mesons produced by EvtGen will be part of the signal chain above. This is not necesarily the case for either the $\overline{D}^*$ or $\overline{D}^0$.

Another use of user decay files is to make a particle stable (if, for example, its decay is uninteresting for your purposes). This is implemented as

```
Decay K0S
Enddecay
```

6 Algorithm

To illustrate how the algorithm works we will first consider an example of the decay

$$B \to D^*(\to D\pi)\tau(\to \pi\nu)\bar{\nu}. \tag{1}$$

The amplitude for this decay can be written as

$$A = \sum_{\lambda_{D^*}\lambda_\tau} A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau} \times A^{D^*\to D\pi}_{\lambda_{D^*}} \times A^{\tau\to\pi\nu}_{\lambda_\tau} \tag{2}$$

Where λ_{D^*} and λ_τ labels the states of the D^* and the τ, respectively, so that

$$A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau} \tag{3}$$

is 6 different amplitudes for the different combinations of D^* τ states. (Note here that the neutrino is treated as a particle with only one state, all neutrinos are assumed to be left-handed and anti neutrinos to be right-handed.) Similarly,

$$A^{D^*\to D\pi}_{\lambda_{D^*}} \text{ and } A^{\tau\to\pi\nu}_{\lambda_\tau} \tag{4}$$

are the amplitudes for a decay of a $D^* \to D\pi$ and $\tau \to \pi\nu$ respectively.

The scheme that was originally implemented in EvtGen was to generate kinematics according to phasespace for the whole decay chain and then calculate that amplitude for the whole decay chain according to Eq. 2. This amplitude was squared to obtain the probability and the event was accepted or rejected based on this probability. This approach works, but has two serious flaws. First, when doing the acceptance rejection you need to know the maximum probability, this gets very hard to keep track of since the number of decay chains get very large. Secondly, as the decay chains get long the efficiency for the acceptance rejection gets low and the time for generating events long.

The source of both these problems are that the decays are treated as a chain and all three decays in the example above are generated at once, in one acceptance rejection. We would like to change the algorithm such that it generates the decay as a sequence of sub-decay. In the example considered here the decay $B \to D^*\tau\nu$ would be done first, then the decay $D^* \to D\pi$ is done. This decay will need to incorporate information about how D^* was produced in the B decay. Further, when the τ is decayed it also has to incorporate the information from the decay of the D^* as well so that correlations between all kinematic variables are correctly generated. This is described bellow for the example considered in this section.

First the decay of the B is considered, kinematics for the decay is generated according to phasespace and the and the probability is then calculated as

$$P_B = \sum_{\lambda_{D^*}\lambda_\tau} |A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau}|^2. \tag{5}$$

Acceptance-rejection is done on the the probability P_B until an event is accepted. After decaying the B we now form the spin-density matrix for the decay of the D^*, in the code this spin-density matrix is refered to as the forward spin-density matrix. This matrix is defined by

$$\rho^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} = \sum_{\lambda_\tau} A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau} A^{B\to D^*\tau\nu *}_{\lambda'_{D^*}\lambda_\tau} \tag{6}$$

The interpretation of this is that here we have summed incoherently over the states of the tau. This means that the spin density matrix ρ^{D^*} describes a D^* in this decay that you have after summing over the degrees of freedom for the tau.

This spin density matrix, ρ^{D^*} is now used to calculate the probability for the D^* decay, again kinematics is generated according to phases pace and the probability is used to accept or reject events based on

$$P_{D^*} = \rho^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} A^{D^*\to D\pi}_{\lambda_{D^*}} A^{D^*\to D\pi *}_{\lambda'_{D^*}} \tag{7}$$

Now we need to decay the D^*; to do this we form the density matrix

$$\tilde{\rho}^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} = A^{D^*\to D\pi}_{\lambda_{D^*}} A^{D^*\to D\pi *}_{\lambda'_{D^*}} \tag{8}$$

This matrix encapsulates the information about the D^* decay that is needed to properly decay the τ with the full correlations between all kinematic variables in the decay. Since this matrix is passed "back" to the decay of the B from the decay of the D^* this matrix is called the backward spin-density matrix in the code.

Using the $\tilde{\rho}^{D^*}$ matrix we can now calculate the forward spin density matrix of the τ

$$\rho^{\tau}_{\lambda_\tau\lambda'_\tau} = \sum_{\lambda_{D^*}\lambda'_{D^*}} \tilde{\rho}^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau} A^{B\to D^*\tau\nu *}_{\lambda'_{D^*}\lambda'_\tau} \tag{9}$$

Note that this spin density matrix for the decay of the τ includes the information about how the D^* decayed through the $\tilde{\rho}^{D^*}$ matrix. Now, as in the other decays, we generate kinematics according to phase space and accepts or rejects based on the probability calculated according to

$$P_\tau = \sum_{\lambda_\tau\lambda'_\tau} \rho^{\tau}_{\lambda_\tau\lambda'_\tau} A^{\tau\to\pi\nu}_{\lambda_\tau} A^{\tau\to\pi\nu *}_{\lambda'_\tau} \tag{10}$$

This algorithm is implemented in the code and it solves two major problems that the original versions of EvtGen had. The algorithm was illustrated above using an example which should convey the idea. I will try to write up the algorithm more formally at some point but the notation for describing it is so awkward.

7 Particle representation

Particles with spin up to spin 2, with the exception of spin 3/2, are handled by classes within the framework of EvtGen. This section will describe how the spin degrees of freedom are represented, and will introduce the classes that represent particles in EvtGen. Table 1 summarizes the different types of particles currently implemented in EvtGen.

Class name	Rep.	J	States	Example
EvtScalar	1	0	1	π, B^0
EvtDirac	u_α	1/2	2	e, τ
EvtNeutrino	u_α	1/2	1	ν_e
EvtVectorParticle	ϵ^μ	1	3	ρ, J/Ψ
EvtPhoton	ϵ^μ	1	2	γ
EvtTensorParticle	$T^{\mu\nu}$	2	5	D_2^*, f_2

Table 1: The different types of particles that supported by EvtGen. The spin 3/2, Rarita-Schwinger, representation has not yet been implemented.

In Table 1, u_α represents a four component Dirac spinor, defined in the Dirac convention for the gamma matrices, as discussed in Section ??. The `EvtDiracParticle` class represents massive spin 1/2 particles that have two (spin) degrees of freedom. Neutrinos are also represented with a 4-component Dirac spinor by the `EvtNeutrinoParticle` class. Neutrinos are assumed to be massless and only left (right) handed (anti-)neutrinos are considered.

ϵ^α represent a complex 4-vector used to represent the spin degrees of freedom for spin 1 particles. Massive spin 1 particles, represented by the `EvtVectorParticle` class, have three degrees of freedom. The `EvtPhotonParticle` class represents massless spin 1 particles, which have only two (longitudinal) degrees of freedom.

Massive spin 2 particles are represented with a complex symmetric rank 2 tensor. These particles are represented by the `EvtTensorParticle` class in EvtGen.

For each particle initialized in EvtGen, a set of basis states are set up. The number of basis states is the same as the number of spin degrees of freedom of the particle. These basis states can be accessed through the `EvtParticle` class in either the particle's rest frame, or in the parent's rest frame. For massless particles only the states in the parents rest frame are available. As an example consider the basis for a massive spin 1 particle in it's own rest frame

$$\epsilon_1^\mu = (0,1,0,0) \tag{11}$$
$$\epsilon_2^\mu = (0,0,1,0) \tag{12}$$
$$\epsilon_3^\mu = (0,0,0,1) \tag{13}$$
$$\tag{14}$$

Note that these basis vectors are mutually orthogonal, and normalized. That is,

$$g_{\mu\nu}\epsilon_i^{*\mu}\epsilon_j^\nu = \delta_{ij}. \tag{15}$$

Further, they form a complete set

$$\sum_i \epsilon_i^{*\mu}\epsilon_i^\nu = g^{\mu\nu} - p^\mu p^\nu/m^2. \tag{16}$$

It is interesting to note here that $g^{\mu\nu} - p^\mu p^\nu / m^2$ is the propagator for an (on-shell) spin 1 particle.

Similarly, each class representing a particle contains a set of basis vectors as the basis for the spin degrees of freedom. These basis vectors must form a complete set, with states that are mutually orthogonal. In programming a model in EvtGen, there is no reason to need to know the exact implementation of these basis states. The code needed to describe the amplitude for a decay process is independent of the basis states, as long as they are complete and orthogonal.

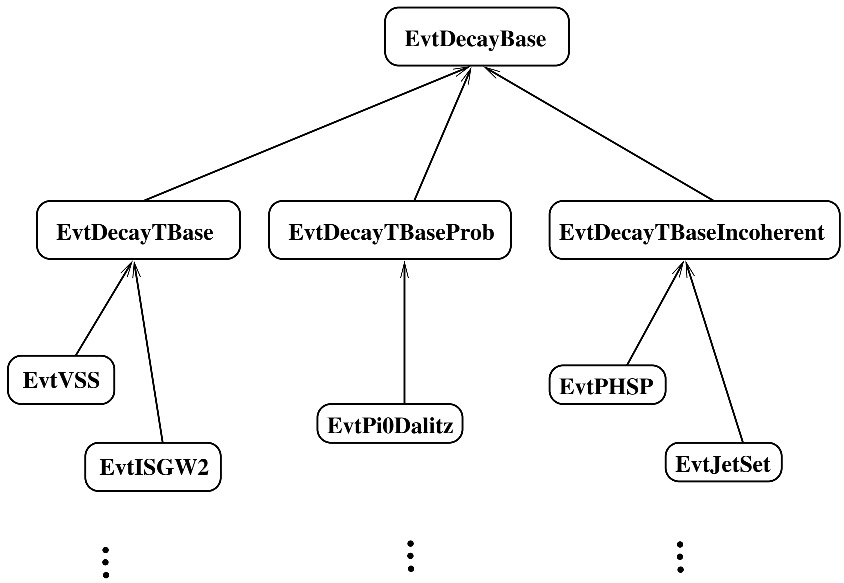

Figure 4: Diagram of the `EvtDecayBase` class and classes derived from it. `EvtDecayTBase`, `EvtDecayTBaseProb`, and `EvtDecayTBaseIncoherent` are templated classes that are used by modules that compute the full decay amplitude, that compute the decay probability, and those that return unpolarized daughters, respectively.

8 Creating a new decay model

EvtGen is organized as a framework in which decays are added as modules, as known as models. This section explains the process of writing modules for newdecay processes.

8.1 Introduction to creating new decay models

Each decay model is a class which is derived from the base class `EvtDecayBase`, as shown in Figure 4. Models may handle many different decays, or might be specialized for one special decay. A model that can handle many different decays is the `VSS` model, which decays a vector meson to a pair of scalar particles. For example, the decays $D^* \to D\pi$ and $\rho \to \pi\pi$ are handled by this model. An example of a highly specialized model is `BTO4PICP`, which describes the decay $B \to \pi^+\pi^-\pi^+\pi^-$.

In writing the decay model there are five member functions that need to be implemented. To illustrate this, we show the `EvtVSS` model as an example. The `EvtVSS` class is defined as:

```
//--------------------------------------------------------------------------
//
// Environment:
//      This software is part of the EvtGen package developed jointly
//      for the BaBar and CLEO collaborations.  If you use all or part
//      of it, please give an appropriate acknowledgement.
//
// Copyright Information:
//      Copyright (C) 1998      Caltech, UCSB
//
```

```
// Module: EvtGen/EvtVSS.hh
//
// Description:
//
// Modification history:
//
//    DJL/RYD     August 11, 1998         Module created
//
//------------------------------------------------------------------------

#ifndef EVTVSS_HH
#define EVTVSS_HH

#include "EvtGen/EvtDecayTBase.hh"
#include "EvtGen/EvtParticle.hh"

class EvtVSS:public  EvtDecayTBase  {

public:

  EvtDecayBase* clone();
  void getName(EvtString& name);

  void Init();
  void InitProbMax();
  void Decay(EvtParticle *p);

};

#endif
```

The simplest function to implement is the `clone()` function. For each decay in the decay table, an instance of the decay model to be used is instantiated. This is done using the prototype pattern, see [1]. Each decay model provides a `clone()` function that creates a new instance of the model. For the `EvtVSS` model, the implementation of `clone()` is

```
EvtDecayBase* EvtVSS::clone(){

  return new EvtVSS;
}
```

The `getName` function specifies the name of the module, to be is used when parsing the decay table. The `EvtReadDecay` class searches for a match between the models in the decay table and the names stored by each implemented module using the `getName` function. For the `EvtVSS` class, the implementation of `getName` is

```
void EvtVSS::getName(EvtString& model_name){

  model_name="VSS";
}
```

The `InitProbMax` member function sets the maximum probability that can be obtained by the model. This maximum probability is used by the EvtGen framework in its decision to keep or reject a decay based on the decay amplitude or probability computed by the model. The maximum probability should be chosen such that it is truely a maximum. However, the larger the maximum probability (as compared to the true maximum), the less efficient the model will be. During the generation of events, if the calculated probability is greater than the maximum probability warning is printed. When particles with finite width are present in the decay tree, it may be difficult to set the maximum probability, as the true maximum probability may depend strongly on the mass of the wide particle. Care should be taken in the implementation of the decay amplitude so that this is not the case. If the true maximum probability does depend strongly on the mass of a particle in the decay, the model will still give correct output, however it will be very inefficient and may get into semi-infinite loops for extreme mass configurations.

The maximum probability is set in `InitProbMax` by calling `SetProbMax(prbmax)`, which is a member function of `EvtDecayBase`.

```
void EvtVSS::InitProbMax() {

   SetProbMax(1.0);
}
```

Models that can handle many different decays might have different maximal probabilities depending on what intial and and final state is beeing generated, therefore it might be neccesary in the member function to set the probability differently depending on what the initial and final states are. The parent and daughter information (`ndaug, parent, narg, daug, args` from the `EvtDecayBase` base class) have been properly filled.

During development of a model it is often convenient not to have to implement the `InitProbMax` function. If this function is not implemented, a default maximum probability is found by calling the model 500 times and using the largest value of the probability in these 500 tries. For a production Monte Carlo, this is not an acceptable way of determining the maximum probability, as during the first call to the model, random numbers will be used up to perform the 500 iterations. This way, the output of the Monte Carlo will depend on the starting event, not just the random number seed. There is a strict requirement on event generators that given a random number seed at the start of the event the same decay must be obtained independently of previous events.

The `Init` is a generic initialization function that will be called once for each decay this model will be used for (once for each occurance in the decay file). The `Init` function allows one to declare member variables that can be precomputed in the `Init` function. An example of such a calculation might be to process the arguments of the model into a more convenient form. For many of the models currently implemented, the `Init` function is used to check of consistency between the number of arguements and the number and types of daughters in the decay table and those expected by the model. There is a virtual `Init` function defined in the `EvtDecayBase` class so that the `Init` member function does not have to be implemented. The `Init` function for the `EvtVSS` class is shown below.

```
void EvtVSS::Init(){

// check that there are 0 arguments.

  if (narg!=0) {

    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " 0 arguments but found:"<<narg<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    assert(0);
  }
```

```
//check that there are 2 daughters.
  if ( ndaug!=2) {
    report(INFO,"EvtGen") << ndaug <<" "<<EvtPDL::name(daug[0])<<endl;
    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " a 2 daughters, found:"<<
                           ndaug<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    assert(0);
  }

//Check the spin types of the parent and daughters.
  EvtSpinType::spintype parenttype = EvtPDL::getSpinType(parent);
  EvtSpinType::spintype d1type=EvtPDL::getSpinType(daug[0]);
  EvtSpinType::spintype d2type=EvtPDL::getSpinType(daug[1]);

  if ( parenttype != EvtSpinType::VECTOR ) {
    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " a VECTOR parent, found:"<<
                           EvtPDL::name(parent)<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    assert(0);
  }
  if ( d1type != EvtSpinType::SCALAR ) {
    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " a SCALAR 1st daughter, found:"<<
                           EvtPDL::name(daug[0])<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    assert(0);
  }
  if ( d2type != EvtSpinType::SCALAR ) {
    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " a SCALAR 2nd daughter, found:"<<
                           EvtPDL::name(daug[1])<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    assert(0);
  }
}
```

Finally, we describe the `Decay` member function. `Decay` does the actual work of decaying the particle. This includes generating kinematics and calculating amplitudes for the process. There is some freedom in how this function is written, some examples are discussed below. For the `EvtVSS` class, the amplitude for the decay of a vector particle into two scalar particles can be written as

$$A = \varepsilon_\mu v^\mu, \tag{17}$$

where ε_μ is the polarization vector for the vector particle and v is the four velocity of the first scalar particle. This amplitude is implemented as

```
void EvtVSS::Decay( EvtParticle *p){

  p->init_daughters(ndaug,daug);

  EvtVector4R pdaug = p->getDaug(0)->getP4();
```

```
  double norm=1.0/pdaug.d3mag();

  vertex(0,norm*pdaug*(p->eps(1)));
  vertex(1,norm*pdaug*(p->eps(2)));
  vertex(2,norm*pdaug*(p->eps(3)));

  return;
}
```

The argument to the `Decay` member function is a pointer to the particle that should be decayed. The first thing that is done is to create the the particles that are to be produced in the decay and to link these particles onto the decay tree. These tasks are performed by the `EvtParticle::init_daughters` member function of the parent particle. The arguments to `init_daughters` are the number and list of daughters, as specified in the decay table. The `init_daughters` function will additionally generate kinematics according to phase space for the daughters.

For models that use the templated class `EvtDecayTBase`, an amplitude for every possible set of set states should be computed. For `EvtVSS`, these amplitudes are calculated according to Eq. 17. These amplitudes are then saved with the decay using the `vertex` member function. The first arguement is the index of the basis vector (starting at 0) used to compute the amplitude. The `EvtDecayTBase` header files shows how this syntax generatizes when more than one nontrival spin is invovled (as is the case for $B \to D^{*}\ell\nu$, for example).

```
void vertex(int i1, const EvtComplex& amp)
void vertex(int i1, int i2, const EvtComplex& amp)
void vertex(int i1, int i2, int i3, const EvtComplex& amp)
void vertex(int *i1, const EvtComplex& amp)
```

After computing the amplitude, the `Decay` function returns and the amplitudes are used in an acceptance-rejection test. If failed the `Decay` routine is called once more and a new kinematic configuration is generated together with new amplitudes.

For models which derive from `EvtDecayTBaseProb` or `EvtDecayTBaseIncoherent`, the `Decay` function is slightly different. As an example of a model derived from the `EvtDecayTBaseProb` class, we show the `PI0DALITZ` model:

```
//--------------------------------------------------------------------------
//
// Environment:
//      This software is part of the EvtGen package developed jointly
//      for the BaBar and CLEO collaborations.  If you use all or part
//      of it, please give an appropriate acknowledgement.
//
// Copyright Information:
//      Copyright (C) 1998      Caltech, UCSB
//
// Module: EvtPi0Dalitz.cc
//
// Description: pi0 -> e+ e- gamma
//
// Modification history:
//
//    DJL/RYD    June 30, 1998          Module created
//
//------------------------------------------------------------------------
```

```
//
#include <fstream.h>
#include <stdio.h>
#include "EvtGen/EvtString.hh"
#include "EvtGen/EvtGenKine.hh"
#include "EvtGen/EvtParticle.hh"
#include "EvtGen/EvtPDL.hh"
#include "EvtGen/EvtReport.hh"
#include "EvtGen/EvtPi0Dalitz.hh"
#include "EvtGen/EvtVector4C.hh"
#include "EvtGen/EvtDiracSpinor.hh"


//code left out

void EvtPi0Dalitz::Decay( EvtParticle *p){

  double prob;
//code to compute the probability is left out

  setProb(prob);

  return;
}
```

Instead of the `vertex` function in `EvtVSS`, the π^0 Dalitz model returns a single probabiltiy using the `setProb` member function.

Finally we show the implementation of the `Decay` member function in the `PHSP` model, which derives from the `EvtDecayTBaseIncoherent` class.

```
void EvtPhsp::Decay( EvtParticle *p ){

//I want to explain the difference between this and
//init_daughters- but I dont see one!!!!!
  p->initializePhaseSpace(ndaug,daug);

  return ;
}
```

In this case, the `Decay` function need only to add initialize the daughters and add them to the particle tree.

In order to write new decay models, it is also necessary to know some of the syntax for finding the momenta and spin basis vector information for a given particle. This information is detailed below in Section 10.

9 Conventions

This Section discusses the conventions that are used for various physics quantities and how they are reperesented in the code.

9.1 Units

Mass, energy and momentum are measured in units of GeV. I.e. in EvtGen, $c = 1$. Similarly, time and space has units of mm.

9.2 Four vectors

There are two types of four vectors used in the EvtGen code: `EvtVector4R` and `EvtVector4C`, they are real and complex respectively.

A four vector is reperesented by $p^\mu = (E, \vec{P})$. When a four vector is used in the code its components are allways corresponding to raised indices. A contractions of two vectors p and k (`p*k`) automatically lowers the indices on k according to the metric $g = \mathrm{diag}(1, -1, -1, -1)$ so that `p*p` is the mass squared of the particle with four momentum p.

9.3 Tensors

The code currently only supports complex rank 2 tensors. As in the case of vectors the tensors are always reperesented with all indices raised. The convention for the totaly antisymetric tensor, $\epsilon_{\alpha\beta\mu\nu}$, is $\epsilon_{0123} = +1$.

9.4 Dirac spinors

The Dirac spinors are reperesented as the 4 component spinors that reperesents initial state fermions or final state anti fermions.

9.5 Gamma matrices

The Dirac gamma matrices are represented in the Dirac-Pauli representation which has

$$\gamma^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad \gamma^1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}, \tag{18}$$

$$\gamma^2 = \begin{bmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ -i & 0 & 0 & 0 \end{bmatrix}, \quad \gamma^3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{19}$$

This gives

$$\gamma^5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \frac{i}{4!}\epsilon_{\lambda\mu\nu\pi}\gamma^\lambda\gamma^\mu\gamma^\nu\gamma^\pi = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{20}$$

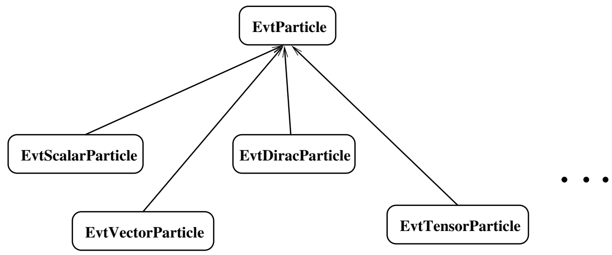

Figure 5: `EvtParticle` forms the base class for the concrete implementations of the different particle types in EvtGen.

10 Classes

This section will discuss the different classes in the EvtGen package except for the classes that makes the models; they are described separately.

10.1 EvtParticle

This is the base class for particles. It contains the common interface to particles such as the four momentum particle number, list of daughters and parent etc. The Particle class forms a base class for the different type of particles: Scalar, Vector, Tensor, Dirac, Photon, and Neutrino, see Figure 5.

The derived classes add a basis for the states used to represent the spin degrees of freedom of the particle. For example the vector particles (`EvtVectorParticle`) adds three complex four-vectors, the massive spin 1/2 particles (`EvtDiracParticle`) adds two Dirac spinors for its basis vectors. This data is stored in the derived class as appropriate for the particle type. The accessor functions are virtual functions in the `EvtParticle` base class. The accessor functions for these basis vectors are shown in Table 2. The default implementation in `EvtParticle` of these accessor functions generates an error message and then aborts the application. The motivation for doing this is that you don't have to do a cast to the derived type to be able to access the basis vectors. But, yes, this is not very good OO practice.

Class	Type	Own rest frame	Parent frame
Scalar	N/A	N/A	N/A
Dirac	`EvtDiracSpinor`	`sp(int i)`	`spParent(int i)`
Neutrino	`EvtDiracSpinor`	N/A	`spNeutrinoParent()`
Vector	`EvtVector4C`	`eps(int i)`	`epsParent(int i)`
Photon	`EvtVector4C`	N/A	`epsParentPhoton(int i)`
Tensor	`EvtTensor4`	`epsTensor`	`epsTensorParent(int i)`

Table 2: This table shows the accessor functions for the basis vectors.

The `init(const EvtId id, const EvtVector4R& p4)` member function sets the id and four-momentum of the particle. This function sets up the basis states of the particle and is hence implemented in the derived class. Note that, in general, to be able to construct the basis vectors both the id and the four-momentum must be know. In the case of a photon, i.e., a massless vector meson, the polarization vectors that makes up the basis state, ε_1 and ε_2, has to be orthogonal to the momentum, i.e. $p \cdot \varepsilon_1 = p \cdot \varepsilon_2 = 0$. For a fermion, represented by a Dirac-spinor, the basis vectors needs to know if it is representing a particle or an anti-particle.

Given a particle there are two member functions that are useful for creating the daughters. These two functions are:

```
void makeDaughters(int ndaug, EvtId *id);
void initializePhaseSpace(int ndaug, EvtId *id);
```

The first of these two routines creates the particles with the types according to the array of `EvtId`'s as specified by `id`. The daughters are added to the parent, i.e., the particle on which this member function is invoked, but the momentum of the particle is not initialized. The second function, `initializePhaseSpace`, is similar except that it generates kinematics according to phase space and initializes the daughters with this momentum.

The `Decay()` member function performs the actual decay of the particle. The `Decay` member function selects a decay by using the class `EvtDecayTable` and then using the `EvtDecayBase` decays the particle. This function recursively decays particles until all particles are stable.

There are various other functions which are useful on `EvtParticle`:

- `EvtParticle* getDaug(int i)`
  Returns a pointer to the n:th daughter of the particle.

- `EvtVector4R getP4Lab()`
  Gets the four-vector of the particle in the laboratory frame. With laboratory frame is meant the rest frame in which the root particles momentum is measured.

- `EvtVector4R get4Pos()`
  Returns the four-position of the particle in the laboratory frame, again the laboratory frame is defined by the position and momentum of the root particle.

- `EvtParticle* getParent()`
  Returns the pointer to the parent particle. If it is the root particle 0 is returned.

- `double mass()`
  Returns the mass of the particle.

- `EvtId getId()`
  Returns the id of the particle.

- `EvtSpinType::spintype getSpinType()`
  Returns the particle type. This is an enum in the `EvtSpinType` class.

- `int getSpinStates()`
  Returns the number of spin degrees of freedom. E.g., 1 for a scalar and 3 for a vector.

- `EvtVector4R& getP4()`
  Returns the four-momentum of the particle in it parents rest frame.

- `void setP4(const EvtVector4R& p4)`
  Set the four-momentum of the particle in its parents rest frame. For a particle that doesn't have a parent its momentum is measured is the laboratory frame.

- `int getNDaug()`
  Returns the number of daughters of the particle.

- `void printTree()`
  Prints out the decay tree starting from the particle the function is invoked on.

- `void printParticle ()`
  Prints detailed information about the particle, this include what type it is and its momentum.

- `void setLifetime(double tau)`
  Sets the lifetime of the particle to `tau`, measured in mm/c.

- `void setLifetime()`
  Generates a lifetime of the particle according to a pure exponential with mean according to `EvtPDT` for the id of the particle.

- `double getLifetime()`
  Returns the lifetime of the particle.

- `void setDiagonalSpinDensity()`
  Initializes the particle with a diagonal spin density matrix. This function is typically invoked on the initial particle before it is decayed.

- `void setVectorSpinDensity()`
  Initializes a vector particle with the appropriate spin density matrix for the production of the vector particle in an e^+e^- interaction.

- `void setSpinDensity(const EvtSpinDensity& rho)`
  Initializes the particle with the spin density matrix `rho`.

Besides these functions there are a few more. These are either considered obsolete and will eventually be removed or should be made private as they are really only helper functions.

10.2 EvtComplex

Currently EvtGen depends on the underlying complex class that is supplied by the compiler. EvtComplex is simply an typedef to the appropriate class name.

10.3 EvtVector4C

This is a class for reperesenting complex four-vectors. Examples of complex four vectors are polarization vectors, ε and currents, e.g., L^μ.

10.4 EvtVector4R

This is a class for representing real four vectors. Examples of real four vectors are momenta and space-time positions.

10.5 EvtTensorC

This class encapsulates the properties of second rank complex tensors. A tensor is reperesented as a 4×4 matrix of complex elements. As in the reperesentation of 4-vectors the tensor elements stored reperesents the tensor $T^{\mu\nu}$, i.e., the tensor with raised indices. The components of a tensor can be manipulated using the `set`, `setdiag`, and `get` member functions. `setdiag` sets the diagonal elements while all other elements are initialized to zero. If `g` is of type `EvtTensorC` tensor `g.setdiag(1.0,-1.0,-1.0,-1.0)` will set `g` to the metric tensor. The member function `trace()` calculates the trace, $T^{\mu}_{\ \mu}$. The operators `+` and `-` are defined to be addition and subtraction of tensors. The operator `*` is defined between a complex number and a tensor and is a scalar multiplication.

A tensor can also be constructed from two four vectors, either EvtVector4C or EvtVector4R, using the direct product function. `T=dir_prod(k,p)` has components $T^{\mu\nu} = k^{\mu}p^{\nu}$. The `dual` function performs a contraction with the totaly anti symmetric tensor $\epsilon^{\mu\nu\alpha\beta}$, `F=dual(T)` has components $F^{\mu\nu} = \epsilon^{\mu\nu\alpha\beta}T_{\alpha\beta}$. The sign convention is $\epsilon_{0123} = +1$. `conj()` takes the complex conjugate of a tensor by conjugating each individual element.

A tensor can also be obtained by contracting two tensors, e.g., $G^{\alpha\beta} = T^{\alpha\mu}F^{\beta}_{\ \mu}$, this is expressed by `G=cont22(T,F)`. The numbers 22 means that it was the second two indices that was contracted. A tensor can also be contracted with a vector. This operation creates a new vector, e.g., `p=T.cont1(k)`, where `cont1` means that the first index of the tensor is contracted with k, $p^{\mu} = T^{\nu\mu}k_{\nu}$.

As special tensors that are defined is the metric tensor, $g = \mathrm{diag}(1, -1, -1, -1)$, which is accessed through the static member function `EvtTensorC::g()`.

The member function `boost(e,px,py,pz)` boost a tensor to the restframe given by (e, p_x, p_y, p_z).

10.6 EvtGammaMatrix

EvtGammaMatrix is a class for handling complex 4×4 matrices. The generic operations that are available are addition (+) and subtractions (-) and multiplaications (*). A EvtGammaMatrix can also be multiplied by a complex scalar. The dump() member function prints out the matrix. Besides these generic operations on matrices there are special operations that are related to gamma matrices. A matrix can be initailized to any of the 4 Dirac γ matrices, $\gamma^0, \gamma^1, \gamma^2, \gamma^3$ or the identity matrix or γ^5. The member functions for doing these operations are g0(), g1(), g2(), g3(), g5(), and id(). Besides these basic initializations there are four more specialized initializations, va0(), va1(), va2(), and va3() that are used for evaluation of the weak lepton current. These matrices are given by $\gamma^{\mu}(1 - \gamma^5)$, where $\mu = (0, 1, 2, 3)$.

There is also an operation (*) defined that multiplies an EvtGammaMatrix with an EvtDiracSpinor and returns an new EvtDiracSpinor. The operation of multiplying two spinors is also defined and has the meaning of first conjugating the first EvtDiracSpinor before performing the multiplication.

10.7 EvtDiracSpinor

The EvtDiracSpinor class encapsulates the properties of a Dirac spinor. It is used to reperesent spin 1/2 particles. EvtDiracSpinor is reperesented as a complex four component spinor. The operations that are available on an EvtDiracSpinor are seting the elements of the spinor, retreving an element printing out the spinor, taking the complex conjugate, boosting the spinor into another Lorentz frame, and evaluating the $V - A$ matrix element. Evaluating the $V - A$ matrix element is done with the function lepton_va_current_d which evaluates the matrix elements

$$L^\mu = \langle \bar{u} | \gamma^\mu (1 - \gamma^5) | v \rangle. \tag{21}$$

where u and v are both EvtDiracSpinors and are supplied as the two arguments to lepton_va_current_d. ($\bar{u} = u^\dagger \gamma^0$ is evaluted by the routine.)

10.8 EvtId

The class `EvtId` is used to particles in EvtGen. It is used, e.g., as argument to the `EvtPDL` member functions to identify the particles.

The `EvtId` class provides the following operators (and a copy constructor) `operator=`, `operator==` and `operator!=`. These are the only operations any user code for implementing particle decays should need. If you find use for other functions you are doing something wrong!

The `EvtId` class contains two member data: `_id` and `_alias`. For regular particles, i.e., non aliases these are the same; an integer from 0 to $n - 1$ where n is the number of particles. These numbers are assigned by `EvtPDL` when reading the particle list. When a particle alias is created, e.g., mypi0, for a pi0. A new `EvtId` is created with the same `_id` as the pi0 but with a unique `_alias`. The two comaprison operators, `==` and `!=`, compares the `_id` when determining if two particles are the same, therefore mypi0 and pi0 are considered the same particles.

There are two places where aliased particles are treated differently than regular particles. The first place is that when decay modes are selected the `_alias` number is used. This is, ofcourse, so that we get different decays for the regular particles and the aliases. This is what we wanted from the begining. The second place where a complication arises is in the definition of the charge conjugate of an aliased particle. For regular particles EvtGen uses the stdhep number to find the charge conjugate. But for aliased particles what the charge conjugate is has to be specified by hand. See discussion section on `EvtPDL`.

10.9 EvtPDL

The particle information read from the `pdt.table` file can be accessed through member functions of the `EvtPDL` class. The following (static) member functions are available:

- `static double getNominalMass(EvtId id)`
  Returns the nominal mass.

- `static double getMass(EvtId id)`
  Generates a mass according to a simple non-relativistic Breit-Wigner shape.

- `static double getMaxMass(EvtId id)`
  The maximal mass that will be generated.

- `static double getMinMass(EvtId id)`
  The minimal mass that will be generated.

- `static double getWidth(EvtId id)`
  The width of the resonance.

- `static double getctau(EvtId id)`
  The lifetime, $c\tau$ (in mm).

- `static int getStdHep(EvtId id)`
  The number according to the stdhep numbering scheme.

- `static int getLundKC(EvtId id)`
  The compact Lund, kc, number.

- `static EvtId evtIdFromStdHep(int stdhep)`
  The `EvtId` given the stdhep number.

- `static EvtId chargeConj(EvtId id)`
  The charge conjugate of the particle.

- `static int chg3(EvtId id)`
  The charge in units of 1/3 of the positron charge.

- `static EvtSpinType::spintype getSpinType(EvtId id)`
  The spin type, i.e., scalar, vector, dirac, neutrino, etc.

- `static EvtId getId(const EvtString& name)`
  Finds the `EvtId` given the name.

- `static EvtString name(EvtId id)`
  The particle name.

- `static void alias(EvtId num,const EvtString& newname)`
  Defines a new particle alias.

- `static void aliasChgConj(EvtId a,EvtId abar)`
  Defines that two aliases are the charge conjugate of each other.

- `static int entries()`
  Number of partilces in the list, not including aliases.

Except for the aliases that `EvtPDL` class is quite simple. The use of alias has been explained previously in the description of the decay tables and in several examples. This class makes use of the `EvtPartProp` class to represent the properties of a single particle.

10.10 EvtPartProp

Class to represent the particle properties of a single particle. Used by `EvtPDL` to keep the particle properties.

10.11 EvtReadDecay

This is a real mess! But it's purpose is to read in the decay table.

10.12 EvtResonance (Kuznetsova)

The EvtResonance class allows one to handle resonances as a single structure. It is currently implemented for decays with three daughter particles. An EvtResonance object is constructed using the four momenta of the parent, the four momenta of the two daughters which constitute a resonance, and the parameters describing the resonance: amplitude c, phase angle θ, width Γ, Breit-Wigner mass m_{BW}, and spin. The spin can be 0, 1, 2, or 3 (as of 09/07/97, it has only been tested for spin 0 and 1, however). The member function resAmpl() returns the complex amplitude for the resonance calculated according to the following formula:

$$c_k e^{i\theta_k} F_k^{BW}(x,y) D_k^{ang}(x,y) \tag{22}$$

where x and y are the invariant masses of the two particle combinations, F_k^{BW} is a normalized Breit-Wigner function, and D_k^{ang} describes the angular distribution for the kth resonance.

More precisely, for spin 0 the (non-relativistic) amplitude is:

$$c\, e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{1}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{23}$$

For spin 1 the amplitude is:

$$c\, e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{cos\phi_3}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{24}$$

For spin 2:

$$c\, e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{\frac{3}{2}cos^2\phi_3-\frac{1}{2}}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{25}$$

For spin 3:

$$c\, e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{\frac{5}{2}cos^3\phi_3-\frac{3}{2}cos\phi_3}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{26}$$

where m12 is the invariant mass of particles 1 and 2, $cos\phi_3$ is the cosine of the angle 3 makes with 2 in the rest frame of 12 (cosine of the angle 3 makes with 1 in the rest frame of 12 is, obviously, $-cos\phi_3$).

Note that if two of the three daughters (for example 2 and 3) are identical, one has to take into account the contributions from two possible combinations 12 and 13, with corresponding signs for the cosines (for spin 1 and higher), and a normalization factor of 1/2.

Another member function, relBrWig(int), returns the relativistic Breit-Wigner amplitude for the $K^*\pi$ (in which case the integer argument should be equal to 1) or $K\rho$ (in which case the argument should be $\neq 1$) resonances. More precisely, for a P-wave decay of a scalar meson (which I'll denote S), the amplitudes are given by:

$$BW(m_{ij}^2) = \frac{\sqrt{\Gamma_0}\, M}{(m_R^2 - m_{ij}^2) - i\Gamma m_R} \tag{27}$$

where the matrix element M is:

$$M = (P_S - P_k)^\mu (g^{\mu\nu} - P_{ij}^\mu P_{ij}^\nu / m_R^2)(P_i^2 - P_j^2) \tag{28}$$

$$|M|^2 = (m_{ik}^2 - m_{jk}^2 - (m_S^2 - m_k^2)(m_i^2 - m_j^2)/m_R^2)\frac{1 + (Rp_{jR})^2}{1 + (Rp_j)^2} \tag{29}$$

$$\Gamma = \Gamma_0 \frac{m_R}{m_{ij}} \left(\frac{p_j}{p_{jR}}\right)^3 \frac{1 + (Rp_{jR})^2}{1 + (Rp_j)^2} \tag{30}$$

Here, m_R and Γ_0 are the mass and width of the m_{ij}^2 resonance; m_S, m_i, m_j, m_k are the masses of the parent and of the i^{th}, j^{th}, k^{th} particles, respectively; p_j is the magnitude of the 3-momentum of the j^{th} particle in the $i-j$ rest frame, and p_{jR} is the same when $m_{ij}^2 = m_R^2$. The value of R for the "centrifugal barrier penetration factor" is taken to be 2 fm for the K^* and 5 fm for the ρ.

10.13 EvtDecayBase

This class is the base class for decay models. It contains the interface for the decay models to the framework. The most important member function is Decay which performs the actual decay of the particle. The member function getName has the single purpose of returning the name of the decay models, this is used when parsing the decay table to identify the correct model. There are two other member functions, Init and InitProbmax. Before explaining what the purpose of these member functions are remember that for each entry in the decay table an instance of the decay model class is created. If the class has any data members that needs to be initialized this should be done in the call to the Init member function. Example of such initialization might be to process the arguments into some more convienient form. InitProbmax is intended for a special kind of initialization, namely of the maximum probability that can be obtained in the decay. This is used by the acceptance rejection method to generate the correct distributions. Note that the call to InitProbmax is done after the call to Init, which means that in InitProbmax you have access to anything calcualted in Init.

10.14 EvtSpinDensity

This class represents spin-density matrices of arbitrary dimensions. (Well, this is not quite true, at the moment it is limited to dimension 5 which is the number of degrees of frsedom of a spin 2 particle.) Functions are provided to manipulate the components of the spin density matrix as well as to calculate probabilites.

10.15 EvtString

This class is used by EvtGen to represent character strings.

10.16 EvtLineShape

This class contains utilities for simualating line shapes of particles. Currently this class only provides the trivial implementation on a non-relativistic Breit-Wigner shape.

10.17 EvtRandom

EvtRandom provides the interface for random numbers that are used in the EvtGen package. We still need to specify the initialization of this class.

10.18 EvtGenKine

EvtGenKine contains tools for generating kinematics such as phase space distributions.

10.19 EvtSymTable

Variables that are defined using a "Define" statement in the decay table are stored in this class. Member functions allow storing new symbols, querring about the existence of variables and looking up their values. This is only used from the `EvtDecayTable` class when reading the decay table.

10.20 EvtAmp2

This class keep tracks of the amplitudes in the decay. It provides member functions to calculate spin-density matrices from the amplitues.

10.21 EvtParser

Used by `EvtDecayTable` to read the decay table. Comments are removed and each token togheter with the current line number is put into a list which is used when building the decay table.

10.22 EvtConst

This class defines useful constants.

```
const double EvtConst::pi            =  3.141592653589793238;
const double EvtConst::twoPi         =  2*pi;
const double EvtConst::radToDegrees =  180./pi;

const double EvtConst::c             =  2.99792458E10; // cm/sec
```

11 CP violation and Mixing

This section discusses how CP violation and mixing work in the generator. These two topics are closely related, since a large fraction of CP violating decays occurs via mixing. Exactly what is the best way of introducing these two effects into EvtGen is still not clear; there are different ways of generating CP violating decays depending on what one is interested in. For example, one might want to generate a sample of events with $B \to a_1^+\pi^-$, and the B's which tag the flavor of this B (e.g., via semileptonic decays) are all B^0's. On the other hand, one might want to generate a sample that has the correct mixture of B^0 and $\bar{B}^0$ tags, as determined by the decay dynamics. From the point of view of implementation these two situations are fairly different, and at the moment we are still working on choosing a consistent way of dealing with them.

This section starts out describing how mixing works, and then proceeds to describe how CP violation is introduced. Examples will be provided. CP violating decays are among the most complicated things this generator does, and various ways in which one might get the generation of CP violating decays wrong will be pointed out.

11.1 Mixing in the $B^0\bar{B}^0$ system.

As is well know, when $\Upsilon(4S)$ decays into the $B^0\bar{B}^0$ system, the B's are produced in a coherent state. This implies that when one of the B's decays, and its flavor is determined, the other B is known to be in a pure state of the opposite flavor. This state then evolves (oscillates) independently, with the oscillation frequency given by $\Delta m/2$, (Δm being $m_2 - m_1$, the mass difference between the two B mass eigenstates). Assuming $|B_0^{\rm phys}(t=0)\rangle = |B^0\rangle$, its time evolution is given by:

$$|B^0_{\rm phys}(t)\rangle = e^{-\Gamma t/2} e^{i\overline{m}t} \left[|B^0\rangle \cos\left(\frac{\Delta m t}{2}\right) + i\frac{q}{p}|\bar{B}^0\rangle \sin\left(\frac{\Delta m t}{2}\right)\right], \tag{31}$$

where $\overline{m} \equiv (m_1+m_2)/2$, $q = e^{i\phi_M}/\sqrt{2}$, and $p = e^{-i\phi_M}/\sqrt{2}$ (ϕ_M is the mixing angle). Time t in this formula is the time between the two B decays.

By projecting $|B_0^{\rm phys}(t)\rangle$ into a pure B^0 or $\bar{B}^0$ state, one can obtain the amplitudes for the B to decay with a given flavor. In turn, squaring them gives the probabilities:

$$P(B^0_{\rm phys} \to B^0) = \frac{1}{2}e^{-\Gamma t}(1+\cos(\Delta m t)) \tag{32}$$

$$P(B^0_{\rm phys} \to \bar{B}^0) = \frac{1}{2}e^{-\Gamma t}(1-\cos(\Delta m t)) \tag{33}$$

$$\tag{34}$$

Integrating these over all times, one obtains the total rates:

$$P(B^0_{\rm phys} \to B^0) = \frac{x_d^2}{2\,(1+x_d^2)} \tag{35}$$

$$P(B^0_{\rm phys} \to \bar{B}^0) = \frac{2+x_d^2}{2\,(1+x_d^2)} \tag{36}$$

$$\tag{37}$$

where $x_d \equiv \frac{\Delta m}{\Gamma}$.

Considering in a similar way the time evolution of the coherently produced state ($\Psi(t=0) = \frac{1}{\sqrt{2}}\left[|B^0\overline{B}^0\rangle - |\overline{B}^0 B^0\rangle\right]$), one can obtain the ratio:

$$\frac{N(\Upsilon(4S) \to B^0 B^0) + N(\Upsilon(4S) \to \bar{B}^0 \bar{B}^0)}{N(\Upsilon(4S) \to B^0 \bar{B}^0)} = \frac{x_d^2}{2 + x_d^2}. \tag{38}$$

Currently, mixing is implemented in the decay model VSS_MIX. This model takes the mass difference to be $0.51 \times 10^{12}\, s^{-1}$, which will be fixed!

One can use this model as follows:

```
Decay Upsilon(4S)
0.5000  B+      B-        VSS;
0.4200  B0      anti-B0   VSS_MIX;
0.0400  B0      B0        VSS_MIX;
0.0400  anti-B0 anti-B0   VSS_MIX;
Enddecay
```

Note that the model itself does *not* determine the fraction of mixed to unmixed events. Instead, the number of mixed and unmixed events is specified by the $\Upsilon(4S)$ branching fractions in the decay table. The VSS_MIX model then generates the correct time evolution for the B's depending on whether the event involves mixing or not.

This implementation will be changed, and the use of the model will be as follows:

```
Decay Upsilon(4S)
0.5000  B+      B-        VSS;
0.5000  B0      anti-B0   VSS_MIX  dm;
Enddecay
```

The new mixing model, VSS_MIX_NEW, will automatically generate events with the right fractions of mixed and un-mixed $\Upsilon(4S)$ decays. Also, the value of Γ (the B width) will be taken from the particle properties, i.e., from the pdt.table in the BaBar environment.

11.2 CP-violation.

The "first generation" of models which were implemented in EvtGen to deal with CP violation did not solve all the problems that are involved in simulating the physics of the $B\bar{B}$ system. To illustrate how these models work, let us consider the following example (the decay $B \to \pi^+\pi^-$):

```
Alias MYB B0
#
Decay Upsilon(4S)
0.70   MYB B0     VSS;
0.30   MYB B0B    VSS;
Enddecay
#
Decay MYB
1.0   pi+ pi-    SSS_CP dm alpha CP |A| arg(A) |Abar| arg(Abar);
Enddecay
End
```

The model used in this example is SSS_CP. Its first argument is dm, the mass difference of the two B mass eigenstates; the second argument, alpha, is the relevant CKM angle (in this case it is α); the third argument, CP, specifies the CP of the final state, and is either $+1$ or -1. The latter argument could in principle be determined from the particle properties, but at the moment EvtGen does not include the C and P quantum numbers in its particle properties list (this should be fixed). The last 4 arguments are the (complex) amplitudes for B^0 and $\bar{B}^0$ to produce the final state.

In this example the $\Upsilon(4S)$ decays with unequal probabilities to "MYB" and B^0 and to "MYB" and $\bar{B}^0$. The B^0 and $\bar{B}^0$ simply decay as a B^0 and a $\bar{B}^0$, respectively, according to the generic decay table. The interesting thing is how "MYB" decays. In the SSS_CP model, "MYB" will produce a $\pi^+\pi^-$ final state, while the other B will provide a tag of the flavor that is listed in the decay of the $\Upsilon(4S)$ (B^0 in 70% of the cases, and $\bar{B}^0$ in 30% of the cases). Therefore, in this example there will be some number of B^0 tags and some number of $\bar{B}^0$ tags.

If one would like to get a sample of events with only B^0 tags, one could do it in the following way:

```
Alias MYB B0
#
Decay Upsilon(4S)
1.00  MYB B0      VSS;
Enddecay
#
Decay MYB
1.0   pi+ pi-     SSS_CP dm alpha CP |A| arg(A) |Abar| arg(Abar);
Enddecay
End
```

Here, the only available tag (as specified by the $\Upsilon(4S)$ branching fraction) is B^0.

When the $\Upsilon(4S)$ is decayed and the two B's are produced, they are assigned lifetimes which are sampled from a pure exponential distribution, with a mean of the average lifetime specified in the particle properties list. (If the decay model is one that includes the effects of mixing, the time distribution is no longer a pure exponential; see the previous section. However, as will be evident below, it doesn't matter what time distribution is used in the decay of the $\Upsilon(4S)$.) Since the lifetime distributions of both the B that decays to the CP channel and the tag B are no longer exponential, the SSS_CP model has to regenerate lifetimes of *both* B's. However, the model does not change what tag one of the B provides. This is important, because it means that this model cannot enforce the right fraction of B^0 and $\bar{B}^0$ tags. **The only way to control this is through the decay table**.

The SSS_CP model handles decays to a pair of scalar particles. Of these, the most important one is probably $B \to \pi\pi$, but the model also works for any other decay with two scalars in the final state, e.g., $K_s\eta'$. Other models which are implemented in a similar way are SVS_CP and STS_CP, which handle the decays to a scalar and a vector and to a scalar and a tensor, respectively. There is also a model for two vectors in the final state: it is called SVV_CP. One complication with this model is that there are three partial waves that contribute to the rate with different CP.

All of these models have one feature in common: they generate the B tags with the relative fractions specified in the decay table. It seems logical that the next step in expanding the functionality of the generator should be implementing some way to generate the right fraction of B^0 and $\bar{B}^0$ tags according to the amplitudes that are specified for a particular decay channel. Indeed, this would be a very natural thing to do, because given the amplitudes, the relative fraction of B^0 and $\bar{B}^0$ tags can be determined. A new class of models which have been recently put into EvtGen does have this feature. These new models are particularly important for decays into final states that are not CP eigenstates, for example $B \to a_1^+\pi^-$, or decays with direct CP violation (e.g., due to the presence of penguins).

Let us consider in more detail what the problem is. The time-dependent CP-asymmetry for the decay $B \to f$, where f is a CP eigenstate, is defined as:

$$A_f(t) = \frac{\Gamma(B^0(t) \to f) - \Gamma(\overline{B}^0(t) \to f)}{\Gamma(B^0(t) \to f) + \Gamma(\overline{B}^0(t) \to f)} \tag{39}$$

If $|A(B^0 \to f)| = |A(\overline{B}^0 \to f)|$, then the integrated asymmetry $\int_{-\infty}^{+\infty} A_f(t)dt = 0$, and the relative fraction of B^0 ($\bar{B}^0$) tags is known (1/2). However, if $|A(B^0 \to f)| \neq |A(\overline{B}^0 \to f)|$ (which can happen, e.g., because of penguin pollution), $\int_{-\infty}^{+\infty} A_f(t)dt \neq 0$, so the fraction of tags must be somehow determined.

One way to get the correct fraction of tags is to use integrated rates. For example, for decays into a CP-eigenstate it can be defined as:

$$fr \equiv \frac{\int_{-\infty}^{\infty} \Gamma(\overline{B}(t)^0 \to f)dt}{\int_{-\infty}^{\infty} \left[\Gamma(B^0(t) \to f) + \Gamma(\overline{B}^0(t) \to f)\right] dt} \tag{40}$$

so that

$$fr = \frac{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right)}{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right) + |A_f|^2 \left(1 + |r_f|^2 + \frac{(1-|r_f|^2)}{1+x_d^2}\right)}, \tag{41}$$

where $x_d \equiv \frac{\Delta m}{\Gamma}$, $r_f = e^{2i\,\phi_M}\,\frac{\overline{A}}{A}$, and $\overline{r}_f = \frac{1}{r_f}$.

Two functions in the new version of EvtOtherB return this fraction (fractB0CP for decays into a CP eigenstate and fractB0nonCP for decays into non-CP eigenstates). However, analytical calculations of the integrals can be done only in the simplest cases. For example, for the $B \to \rho\pi$ decay one would have to integrate over the Dalitz plot as well as over time to get the total rates, which is very hard to do! A better solution is to use the so-called acceptance-rejection method, which is effectively equivalent to numerical integration. This method has been used in the "second generation" models which introduce direct CP-violation into EvtGen.

The new models that are available are: `SSS_CP_PNG` for the $B \to \pi^+\pi^-$ decay with penguin contributions; `BTO2PI_CP_ISO` for the three isospin-related $B \to \pi\pi$ modes; `BTOKPI_CP_ISO` for the four isospin-related $B \to K\pi$ modes; and `SVS_CP_ISO` for the five isospin-related generic $scalar \to (pseudo)scalar + vector$ modes. There is another decay model for the latter case, which also returns the correct fraction of B^0 ($\bar{B}^0$) tags. It is called `SVS_NONCPEIGEN`. Here is an example of how this model can be used for the $B \to a_1\pi$ decay:

```
Alias MYB B0
Decay Upsilon(4S)
1.000 MYB anti-B0                          VSS;
Enddecay
Decay MYB
```

```
1.000 a_1-  pi+          SVS_NONCPEIGEN 1.22 0.51e12 1.0 0.0
                                                     1.0 0.0 3.0 0.1
                                                     3.0 0.1 1.0 0.0;
Enddecay
End
```

The first parameter of this model is the corresponding CKM angle (in this case, it is α); the second parameter is Δm; the third parameter is the CP of the final state; the fouth parameter is the "flip" which determines the fraction of "MYB" $\to f$ to "MYB" $\to \overline{f}$ decays, where the state specified in the decay table is considered the "f" state. If the flip is set to 0, "MYB" always decays into the f state. The next four parameters are the absolute values and phases for the amplitudes $B^0 \to f$ and $\bar{B}^0 \to f$, respectively, and the last four parameters are those for the amplitudes $B^0 \to \overline{f}$ and $\overline{B}^0 \to \overline{f}$.

It is important not to confuse the "flip" parameter with the number of tags. For example, one would like to generate 10,000 $B\overline{B}$ events with the flip parameter set to 0.5. This means that one would get 5,000 "MYB" $\to f$ decays and 5,000 "MYB" $\to \overline{f}$ decays, but within each 5,000 event sample the number of B^0 and $\bar{B}^0$ tags is determined by the amplitudes specified in the decay file!

As before, "MYB" in this example decays into a CP-mode, while the other B provides a tag. However, the model itself generates the correct fraction of B^0 ($\bar{B}^0$) tags, which is determined from the last eight parameters (the amplitudes), and is independent of the $\Upsilon(4S)$ branching fractions.

11.3 Special CP models

The models described above are meant to be generic models that can be applied to different decays with appropriate arguments. All of the above models are two body decays. There is, however, many other decays that are not two body decays and these can not be treated quite as simply since there will in general be interference between various resonances. As an example consider the decay $B \to \pi^+\pi^-\pi^0$ which is dominated by the $\rho(770)$ resonances.

To handle the $B \to 3\pi$ and the $B \to 4\pi$ decays two special models have been written. These models are called `BTO3PI_CP` and `BTO4PI_CP`. The second of model `BTO4PI_CP` needs more work interms of understanding the conventions of the phases between the resonances that contributes.

12 Semileptonic decays in EvtGen

12.1 Introduction

We summarize the treatment of semileptonic decays in EvtGen. A framework is setup such that the implementation of a model for semileptonic form factors requires only the coding of the form factors themselves. Currently, we consider semileptonic decays from pseudoscalar mesons into pseudoscalar, vector, or tensor meson daughters. In section 12.2 we define the conventions used in the implementation of semileptonic decays. Section 12.3 describes the framework from which semileptonic form factor models can be built. An example of such a model is given in section 12.4. Finally, we list the form factor models implemented in EvtGen in section 12.5.

12.2 Semileptonic conventions

We have tried to follow standard descriptions of semileptonic form factors in EvtGen. For decays of pseudoscalar mesons to pseudoscalar daughters, we describe the form factors as

$$\begin{aligned} < S(k)|V_\mu|P(p) > &= \left\{(p+k)_\mu - \frac{m_P^2 - m_S^2}{q^2} q_\mu\right\} f_+(q^2) \\ &+ \left\{\frac{m_P^2 - m_S^2}{q^2} q_\mu\right\} f_0(q^2) \end{aligned} \quad (42)$$

P denotes the parent (pseudoscalar) meson (B, D or B_s) and S denotes a pseudoscalar daughter (D, K, or π). In this form, we have the constraint $f_+(0) = f_0(0)$.

For decays to vector mesons, we use the form

$$\begin{aligned} < V(k)|(V-A)_\mu|P(p) > &= \epsilon^*_\mu (m_P + m_V) \mathrm{A}_1(q^2) - (p+k)_\mu (\epsilon^* p) \frac{\mathrm{A}_2(q^2)}{(m_P + m_V)} \\ &- q_\mu (\epsilon^* p) \frac{2 m_V}{q^2} (\mathrm{A}_3(q^2) - i\mathrm{A}_0(q^2)) \\ &+ \epsilon_{\mu\nu\rho\sigma} \epsilon^{*\nu} p^\rho k^\sigma \frac{2\mathrm{V}(q^2)}{m_P + m_V} \end{aligned} \quad (43)$$

Where V denotes a vector daughter meson (D^*, K^* or ρ) and A_3 is defined as

$$\mathrm{A}_3(q^2) = \frac{m_P + m_V}{2m_V} \mathrm{A}_1(q^2) - \frac{m_P - m_V}{2m_V} \mathrm{A}_2(q^2) \quad (44)$$

where $\mathrm{A}_0(0) = \mathrm{A}_3(0)$.

For semileptonic decays to tensor mesons (D_2^*), to be denoted as T, we use the form

$$\begin{aligned} < T(k)|(V-A)_\mu|P(p) > &= ih(q^2) \epsilon_{\mu\nu\lambda\rho} \epsilon^{*\nu\alpha} p_\alpha (p+k)^\lambda (p-k)^\rho - k(q^2) \epsilon^*_{\mu\nu} p^\nu \\ &+ b_+(q^2) \epsilon^*_{\alpha\beta} p^\alpha p^\beta (p+k)_\mu - b_-(q^2) \epsilon^*_{\alpha\beta} p^\alpha p^\beta (p-k)_\mu \end{aligned} \quad (45)$$

12.3 Semileptonic framework

There are two abstract classes responsible for the semileptonic decay framework. The first of these is EvtSemiLeptonicFF, which provides the framework for the calculations of form factors. There are three member functions:

- ```
  virtual void getscalarff( int parent, int daught,
                        double t, double mass, double *fpf,
                        double *fmf )
  ```
- ```
  virtual void getvectorff( int parent, int daught,
                        double t, double mass, double *a1f,
                        double *a2f, double *vf, double *a0f )
  ```
- ```
  virtual void gettensorff( int parent, int daught,
                        double t, double mass, double *hf,
                        double *kf, double *bpf, double *bmf )
  ```

All form factors models should include a class derived from EvtSemiLeptonicFF, which will implement the relevant member functions of EvtSemiLeptonicFF.

These form factor implementations are used by classes derived from the abstract class EvtSemiLeptonicAmp, which contains the member function:

- ```
  virtual void CalcAmp( EvtParticle *parent,
           EvtSemiLeptonicFF *FormFactors  )
  ```

Classes to calculate the amplitude for semileptonic pseudoscalar to pseudoscalar decays, pseudoscalar to vector decays, and pseudoscalar to tensor decays are EvtSemiLeptonicScalarAmp, EvtSemiLeptonicVectorAmp, and EvtSemiLeptonicTensorAmp, respectively. These routines are passed form factors which must follow the conventions described in section 12.2. An example of how to use this framework is presented in section 12.4.

12.4 Examples

Above we have defined the conventions followed in the standard semileptonic framework of EvtGen. Here we show how new form factor models can be easily implemented. We take as an example a heavy quark effective theory model of $B \to D^* \ell\nu$ form factors, where ℓ is either an electron or muon. In this decay we can write the form factors as:

$$A_1 = \frac{h_{A_1}(w)}{R^*}\Big(1 - \frac{q^2}{(M_B + M_{D^*})}\Big) \tag{46}$$

$$V = R_1 \frac{h_{A_1}(w)}{R^*} \tag{47}$$

$$A_2 = R_2 \frac{h_{A_1}(w)}{R^*} \tag{48}$$

where

$$h_{A_1}(w) = h_{A_1}(1)\Big(1 - \rho^2_{A_1}(w-1)\Big), \tag{49}$$

$$R^* = \frac{2.0(M_B M_{D^*})^{1/2}}{(M_B + M_{D^*})}, \tag{50}$$

and

$$w = \frac{M_B^2 + M_{D^*}^2 - q^2}{2 M_B M_{D^*}} \tag{51}$$

To implement a decay model using these form factors we must write two classes

- ```
  class EvtHQETFF : public EvtSemiLeptonicFF {

  public:
    EvtHQETFF(double hqetrho2, double hqetr1, double hqetr2);

    void getvectorff( int parent, int daught,
                          double t, double mass, double *a1f,
                          double *a2f, double *vf, double *a0f );

  private:
    double r1;
    double rho2;
    double r2;
  };
  ```

- ```
  class EvtHQET:public  EvtDecayTBase<EvtHQET>  {

  public:

    static void getName(char *name,int &narg);

    void Decay(EvtParticle *p);
    void InitProbMax();
    void Init();

  private:
    EvtSemiLeptonicFF *hqetffmodel;
    EvtSemiLeptonicAmp *calcamp;
  };
  ```

This class is to implement $B \to D^* \ell \nu$ decays, so only the vector form factor routine must be provided. The constuctor allows R1, R2 and ρ^2 to be passed in as a run time arguement. We will discuss this point more below. The implementation of `EvtHQETFF` is quite simple:

```
EvtHQETFF::EvtHQETFF(double hqetrho2, double hqetr1, double hqetr2) {

  rho2 = hqetrho2;
  r1 = hqetr1;
  r2 = hqetr2;
  return;
}
```

```
void EvtHQETFF::getvectorff( int parent, int daught,
                       double t, double mass, double *a1f,
                             double *a2f, double *vf, double *a0f ){


  double mb=EvtPDL::nom_mass(parent);
  double w = ((mb*mb)+(mass*mass)-t)/(2.0*mb*mass);

// Form factors have a general form, with parameters passed in
// from the arguements.

  double rstar = ( 2.0*sqrt(mb*mass))/(mb+mass);
  double ha1 = 1-rho2*(w-1);

  *a1f = (1.0 - (t/((mb+mass)*(mb+mass))))*ha1;
  *a1f = (*a1f)/rstar;
  *a2f = (r2/rstar)*ha1;
  *vf = (r1/rstar)*ha1;
  *a0f = 0.0;

  return;
 }
```

The `getvectorff` routine takes the parent meson, daughter vector meson, q^2 (`t`), and the daughter meson mass. The form factors A_1, A_2, V and A_0 are returned.

The `EvtHQET` class is also quite easy to implement. As with other classes which handle decay models, it is derived from `EvtDecayTBase`, which is described elsewhere. The class should provide a `Decay` routine as well as `getName`, `Init`, `clone` and `InitProbMax` routines. Additionally, the form factor model and amplitude calculation implementations are handled as memeber data. The `EvtHQET` implementation is:

```
void EvtHQET::getName(char *model_name,int &narg){

  strcpy(model_name,"HQET");
}

EvtDecayBase* EvtHQET::clone(){

  return new EvtHQET;

}


void EvtHQET::Decay( EvtParticle *p ){

  p->init_daughters(ndaug,daug);
```

```
  calcamp->CalcAmp(p,hqetffmodel);
}

void EvtHQET::InitProbMax() {

  SetProbMax(20000.0);
}

void EvtHQET::Init(){

  if ( ndaug!=3 ) {
     report(ERROR,"EvtGen") << "Wrong number of daughters in EvtHQET.cc\n";
  }

  hqetffmodel = new EvtHQETFF(args[0],args[1],args[2]);
  calcamp = new EvtSemiLeptonicVectorAmp;
}
```

This class would be virtually unchanged for different form factor models. In `Decay`, after the initialization of the daughter particles, `CalcAmp` (in this case from `EvtSemiLeptonicVectorAmp` is called. If the standard form factor parameterizations are used, this routine does not need to be rewriten. The `Init` function is responsible for two things. First the correct form factor model must be initialized. As `EvtHQETFF` takes arguements, these must be passed into the constructor. The arguements are otherwise private to the `EvtHQET` model. Second, the `CalcAmp` function must be properly initialized such that the amplitude is calculated for vector daughter mesons. For models with form factors for daughter mesons of different spins, the meson spin (`EvtPDL::spin(daug[0])` should be used to decide how to initialize `calcamp`.

With the existing framework, the implementation of new form factor models should be quite simple. The majority of the work is in the coding of the form factors themselves

12.5 Available models

There several semileptonic form factor models implemented within EvtGen and the `EvtSemiLeptonicAmp` framework. In this section we describe the basic features of each, including the modes available in each model and what arguements must be passed in. This is by no means a complete list of all possible semileptonic models, and the authors would welcome any additional form factor models implemented.

12.5.1 EvtSLPole

- A general pole form is implemented for each form factor, using the form

$$F = \frac{F(0)}{(1.0 + a\frac{q^2}{M^2} + b\frac{q^4}{M^4})^p} \tag{52}$$

  where M is the mass of the parent meson.

- Leptons available: e, μ, and τ.

- Arguements: For each form factor there are four arguements, $F(0)$, a, b, and p. Note that p can be a non-integer.
  - For scalar daughters the arguements should be ordered as $F(0)_{f_+}$ a_{f_+} b_{f_+} p_{f_+} $F(0)_{f_0}$ a_{f_0} $b_{f_0} p_{f_0}$.
  - For vector daughters the arguements should be ordered as $F(0)_{A_1}$ a_{A_1} b_{A_1} p_{A_1} $F(0)_{A_2}$ a_{A_2} b_{A_2} p_{A_2} $F(0)_V$ a_V b_V p_V $F(0)_{A_0}$ a_{A_0} b_{A_0} p_{A_0}
  - For tensor daughters the arguements should be ordered as $F(0)_k$ a_k b_k p_k $F(0)_h$ a_h b_h p_h $F(0)_{b_+}$ a_{b_+} b_{b_+} p_{b_+} $F(0)_{b_-}$ a_{b_-} b_{b_-} p_{b_-}
- In DECAY.DEC

```
0.0210   D+     e-   anti-nu_e          SLPOLE .... arguements;
```

12.5.2 EvtISGW2

- Implementation of form factors defined from Scora and Isgur [2].
- Leptons available: e, μ, and τ.
- Arguements: None
- In DECAY.DEC

```
0.0210   D+     e-   anti-nu_e          ISGW2;
```

We have implemented decays of B,D,D_s, and B_s mesons as described by Scora and Isgur.

12.5.3 EvtISGW

- Implementation of form factors defined from Isgur, Scora, Grinstein, and Wise [3].
- Leptons available: e and μ only.
- Arguements: None
- In DECAY.DEC

```
0.0210   D+     e-   anti-nu_e          ISGW;
```

12.5.4 EvtHQET

- Simple implementation of form factors within HQET using a linear form for ρ^2. This model is described in more detail above.
- Leptons available: e and μ only.
- Arguements:ρ^2, $R1$, and $R2$.
- In DECAY.DEC

```
    0.050   D*+     e-   anti-nu_e          HQET 0.92 1.18 0.72;
```

This model is most useful for the decay $B \to D^* \ell \nu$, however no assumptions are made about either the parent meson or the daughter meson. Any pseudoscalar semileptonic decay to a vector meson could be modeled using these form factors.

A Decay models

This section lists the different decay models that has been implemented in EvtGen. It is strongly urged that all decays that are implemented are added to this list with, at least, a minimal description if what they are doing.

The models are organized in alphabetical order here. Each model is briefly described with respect to what decays it can handle and what the arguments mean and one or more examples are given. For further examples of use please see the decay table, `DECAY.DEC`

A.1 BHADRONIC

Author: Ryd
Usage:
`BrFr P1 P2 ...  PN BHADRONIC JH JW;`

Explanation:
This is an experimental model for hadronic B decays. Until further developed this is not recommended to be used. For questions ask Anders Ryd.

A.2 BTO3PI_CP

Author: Le Diberder, Versille
Usage:
`BrFr P1 P2 P3 BTO3PI_CP dm alpha;`

Explanation:

This model is for neutral B decays in $\pi^-\pi^+\pi^0$. Several resonances are taken into account: the $\rho(770)$, the $\rho(1450)$, and the $\rho(1700)$, the generator therefore implements the interferences between all different final states (e.g. $B^0 \to \rho^+(770)\pi^- \to \pi^+\pi^0\pi^-$ and $B^0 \to \rho^-(770)\pi^+ \to \pi^-\pi^0\pi^+$). Several Breit-Wigners descriptions are available. By default, the generator is initialized with relativistic Breit-Wigners according to the Kuhn-Santamaria model where the parameters have been fitted by Aleph with e^+e^- and $\tau^+\tau^-$ data [4]. It uses a pole-compensation method to generate the events efficiently by taking into account the poles due to the Breit-Wigners of the ρ's [5].

The generator returns the amplitudes for $B^0 \to 3\pi$, and $\overline{B^0} \to 3\pi$ for the kinematics of the generated final state. It makes use of values of Tree and Penguins amplitudes and phases which have been computed by the LPTHE using the factorization approximation and the Orsay quark model, on the basis of the Isospin relation (no ElectroWeak Penguins are included here, and the strong phases are set to zero).The $\rho^0\pi^0$ gets a very low branching ratio due to color-suppression [6].

Example:
The example shows how to generate $B^0 \to \pi^-\pi^+\pi^0$.

```
Decay B0
1.000 pi- pi+ pi0   BTO3PI_CP   dm alpha;
```

```
Enddecay
```

Notes:
This routine makes use of a fortran routine to perform the actual calculation of the amplitude.

A.3 CB3PI-MPP

Author: Le Diberder, Versillé
Usage:
`BrFr P1 P2 P3 CB3PI-MPP dm alpha;`

Explanation:
This model is for charged B decays in $\pi^\pm\pi^+\pi^-$. It is built on the same basis than the BTO3PI_CP model, making also use of interferences between the three ρ bands: $\rho(770)$, the $\rho(1450)$, and the $\rho(1700)$. The amplitudes are computed by the LPTHE.

Example:
The example shows how to generate $B^+ \to \pi^+\pi^+\pi^-$.

```
Decay B+
1.000 pi+ pi+ pi-   CB3PI-MPP   dm alpha;
Enddecay
```

Notes:
This routine makes use of a fortran routine to perform the actual calculation of the amplitude.

A.4 CB3PI-P00

Author: Le Diberder, Versillé
Usage:
`BrFr P1 P2 P3 CB3PI-P00 dm alpha;`

Explanation:
This model is for charged B decays in $\pi^\pm\pi^0\pi^0$. It is built on the same basis than the BTO3PI_CP model, making also use of interferences between the three ρ bands: $\rho(770)$, the $\rho(1450)$, and the $\rho(1700)$. The amplitudes are computed by the LPTHE.

Example:
The example shows how to generate $B^+ \to \pi^+\pi^0\pi^0$.

```
Decay B+
1.000 pi+ pi0 pi0   CB3PI-P00   dm alpha;
Enddecay
```

Notes:
This routine makes use of a fortran routine to perform the actual calculation of the amplitude.

A.5 BTOKPIPI_CP

Author: Le Diberder, Versillé
Usage:
`BrFr P1 P2 P3 BTOKPIPI_CP dm alpha;`

Explanation:
This model is for neutral B decays in $K\pi\pi$ (note that the B^0 decays in $K^+\pi^-\pi^0$ and the $\overline{B^0}$ in $K^-\pi^+\pi^0$). It generates interferences between different resonances:

- $B^0 \to K^{*+}\pi^-$, with $K^{*+} \to K^+\pi^0$,
- $B^0 \to K^{*0}\pi^0$, with $K^{*0} \to K^+\pi^-$,
- $B^0 \to K^-\rho^+$, with $\rho^+ \to \pi^+\pi^0$

It also provides the amplitudes for the CP-conjugate channels $\overline{B^0} \to K^-\pi^+\pi^0$. The Tree and Penguins amplitudes are computed by the LPTHE.

Example:
The example shows how to generate $B^0 \to K^+\pi^-\pi^0$.

```
Decay B0
1.000 K+ pi- pi0    BTOKPIPI_CP dm alpha;
Enddecay
```

Notes:
This routine makes use of a fortran routine to perform the actual calculation of the amplitudes.

A.6 BTO4PI_CP

Author: Ryd
Usage:
`BrFr P1 P2 P3 P4 BTO4PI_CP dm alpha +8 amplitudes;`

Explanation:
This model is for $B \to \pi^+\pi^-\pi^+\pi^-$. It implements the time dependence of the decay correctly depending on where in the dalitz plot you are. The amplitudes that needs to be specified are $B \to a_1^+\pi^-$, $\bar{B} \to a_1^+\pi^-$, $B \to a_2^+\pi^-$, $\bar{B} \to a_2^+\pi^-$, $B \to a_1^-\pi^+$, $\bar{B} \to a_1^-\pi^+$, $B \to a_2^-\pi^+$, and $\bar{B} \to a_2^-\pi^+$.

Example:
The example shows how to generate $B^0 \to \pi^+\pi^-\pi^+\pi^-$.

```
Decay B0
1.000 pi+ pi- pi+ pi-   BTP4PI_CP   dm alpha 1.0 0.0 1.0 0.0
                                              1.0 0.0 1.0 0.0
                                              1.0 0.0 1.0 0.0
                                              1.0 0.0 1.0 0.0;
Enddecay
```

Notes:
This routine is still developing.

A.7 BTO2PI_CP_ISO

Author: NK
Usage:
`BrFr P1 P2 BTO2PI_CP_ISO beta dm` $|A_2|\ \varphi_{A_2}\ |\overline{A}_2|\ \varphi_{\overline{A}_2}\ |A_0|\ \varphi_{A_0}\ |\overline{A}_0|\ \varphi_{\overline{A}_0}$;

Explanation:
This model approaches the three $B \to \pi\pi$ modes from the point of view of isospin analysis. It is applicable to both the two B^0 ($\overline{B}^0$) modes, in which case it takes into account mixing, and to the B^+ (B^-) mode, as all three modes should indeed be treated together in this approach. Following the conventions of Lipkin, Nir, Quinn, and Snyder (Phys. Rev. D**44**, 1454 (1991)), the various decay amplitudes can be written as follows:

$$A(B^+ \to \pi^+\pi^0) \equiv A^{+0} = 3\,A_2 \tag{53}$$

$$A(B^0 \to \pi^+\pi^-) \equiv \sqrt{\frac{1}{2}}\,A^{+-} = A_2\, -\, A_0 \tag{54}$$

$$A(B^0 \to \pi^0\pi^0) \equiv A^{00} = 2\,A_2\, +\, A_0, \tag{55}$$

where A_2 is the amplitude for $I_f = 2$ states (tree only), and A_0, for $I_f = 0$ states (where both tree and penguin contribute).

The model's parameters are:

- beta = corresponding CKM angle
- dm = $B^0\overline{B}^0$ mass difference ($\approx 0.5 \times 10^{12} s^{-1}$).
- $|A_2|$, φ_{A_2} = magnitude and phase of the corresponding amplitude
- $|\overline{A}_2|$, $\varphi_{\overline{A}_2}$ = magnitude and phase of the amplitude for the CP-conjugate process
- $|A_0|$, φ_{A_0} = magnitude and phase of the corresponding amplitude
- $|\overline{A}_0|$, $\varphi_{\overline{A}_0}$ = magnitude and phase of the amplitude for the CP-conjugate process

Example:

```
Decay B0
1.000 pi+ pi-  BTO2PI_CP_ISO  beta dm 1.0 gamma 1.0 -gamma
                                      1.0 gamma 1.0 -gamma;
Enddecay
```

Notes:
Precise numerical estimates for the amplitudes are not available at the moment.

A.8 BTOKPI_CP_ISO

Author: NK
Usage:
BrFr PI K BTOKPI_CP_ISO beta dm $|U|$ φ_U $|\overline{U}|$ $\varphi_{\overline{U}}$ $|V|$ φ_V $|\overline{V}|$ $\varphi_{\overline{V}}$ $|W|$ φ_W $|\overline{W}|$ $\varphi_{\overline{W}}$;

Explanation:
This model considers the four $B \to \pi K$ modes from the point of view of isospin analysis. It is applicable to both the two B^0 ($\overline{B}^0$) modes and to the two B^+ (B^-) modes, as all four modes should indeed be treated together in this approach. Following the conventions of Lipkin, Nir, Quinn, and Snyder (Phys. Rev. D**44**, 1454 (1991)), the various decay amplitudes can be written as follows:

$$A(B^+ \to \pi^0 K^+) \equiv A^{0+} = U - W \tag{56}$$

$$A(B^+ \to \pi^+ K^0) \equiv \sqrt{\frac{1}{2}} A^{+0} = V + W \tag{57}$$

$$A(B^0 \to \pi^- K^+) \equiv \sqrt{\frac{1}{2}} A^{-+} = V - W \tag{58}$$

$$A(B^0 \to \pi^0 K^0) \equiv A^{00} = U + W, \tag{59}$$

where W, U, and V are linear combinations of the three independent amplitudes A_{I_t,I_f} for various transition (I_t) and final (I_f) isospins (please see the reference for more details). Note that both U and V are tree-only amplitudes, whereas W includes both tree and penguin contributions.

The model's parameters are:

- beta = corresponding CKM angle
- dm = $B^0\overline{B}^0$ mass difference ($\approx 0.5 \times 10^{12} s^{-1}$).
- $|U|$, φ_U = magnitude and phase of the corresponding amplitude
- $|\overline{U}|$, $\varphi_{\overline{U}}$ = magnitude and phase of the amplitude for the CP-conjugate process
- $|V|$, φ_V = magnitude and phase of the corresponding amplitude
- $|\overline{V}|$, $\varphi_{\overline{V}}$ = magnitude and phase of the amplitude for the CP-conjugate process
- $|W|$, φ_W = magnitude and phase of the corresponding amplitude
- $|\overline{W}|$, $\varphi_{\overline{W}}$ = magnitude and phase of the amplitude for the CP-conjugate process

Example:

```
Decay B0
1.000  K+ pi-  BTOKPI_CP_ISO   beta dm  1.0 gamma 1.0 -gamma
                                        1.0 gamma 1.0 -gamma
                                        1.0 gamma 1.0 -gamma;

Enddecay
```

Notes:
Precise numerical estimates for the amplitudes are not available at the moment.

A.9 D_DALITZ;

Author: Kuznetsova
Usage:
`BrFr D1 D2 D3 D_DALITZ;`

Explanation:
The Dalitz amplitude for three-body $K\pi\pi$ D decays; namely, for decays

- $D^+ \to K^-\pi^+\pi^+$ or $D^- \to K^+\pi^-\pi^-$,
  with the resonances (for the D^+ mode) $\overline{K}^*(892)^0\pi^+$, $\overline{K}^*(1430)^0\pi^+$, and $\overline{K}^*(1680)^0\pi^+$, using data from the E691 Fermilab experiment [7].

- $D^+ \to \overline{K}^0\pi^+\pi^0$ or $D^- \to K^0\pi^-\pi^0$,
  with the resonances (for the D^+ mode) $\overline{K}^*(892)^0\pi^+$, and $\overline{K}^0\rho^+$, using data from MARK III [8].

- $D^0 \to \overline{K}^0\,\pi^+\pi^-$ or $\overline{D}^0 \to K^0\,\pi^-\pi^+$,
  with the resonances (for the D^0 mode) $K^*(892)^-\pi+$ and $\overline{K}^0\rho(770)^0$, using data from [7].

- $D^0 \to K^-\pi^+\pi^0$ or $\overline{D}^0 \to K^+\pi^-\pi^0$,
  with the resonances (for the D^0 mode) $\overline{K}^*(892)^0\pi^0$, $K^*(892)^-\pi^+$, and $K^-\rho(770)^+$, using data from [7].

Be aware that the $D^+ \to \overline{K}^0\pi^+\pi^0$ and $D^- \to K^0\pi^-\pi^0$ modes currently use the results from Mark III [8], which are based on rather limited statistics.

Example:
To generate the decay $D^+ \to K^-\pi^+\pi^+$ the following entry in the decay table should be used

```
Decay D+
1.000 K- pi+ pi+      D_DALITZ;
Enddecay
```

Notes:
The order in which the particles are listed is very important: the kaon should always be first, and for the modes with the neutral pion the π^0 should always be last.

A.10 GOITY_ROBERTS

Author: Alain,Ryd
Usage:
`BrFr M1 M2 L N GOITY_ROBERTS ;`

Explanation:
Model for the non-resonant $D^{(*)}\pi\ell\nu$ decays of B mesons. The daughters are in the order: D-meson, pion, lepton and last the neutrino.

Example:

```
Decay B0
1.000 D0B  pi- e+ nu_e                    GOITY_ROBERTS;
Enddecay
```

Notes:
This is not exactly what was published by Goity and Roberts, partly due to errors in the paper and because the D^* had to be removed from the $D\pi$ non-resonant.

A.11 HQET

Author: Lange
Usage:
`BrFr M L N HQET RHO2 R1 R2;`

Explanation:
Model for the $D^*\ell\nu$ decay of B mesons according to a HQET inspired parameterization. The daughters are in the order: D^*, lepton and last the neutrino. Since only the three form factors that contributes in the zero lepton mass limit are included the model is not accurate for τ's. The arguments, `RHO2`, `R1`, and `R2` are the form factor slope, $\rho^2_{A_1}$ and the form factor ratios R_1 and R_2 respectively. These are defined, and measured, in [9]

Example:
Decay of $B^0 \to D^*\ell\nu$ using HQET model

```
Decay B0
1.000 D*-  e+ nu_e               HQET3S1 0.92 1.18 0.72;
Enddecay
```

Notes:
The values in the example above comes from the measurement in [9] by the CLEO collaboration.

A.12 ISGW

Author: Lange, Ryd
Usage:
`BrFr D1 D2 D3 ISGW ;`

Explanation:
This is a model for semileptonic decays of B, and D mesons according to the ISGW model [3]. The first daughter is the meson produced in the semileptonic decay. The second and third

argument is the lepton and the neutrino respectively. See Section 12 for more details about semileptonic decays.

Example:

The example shows how to generate $\bar{B}^0 \to D^{*+} e\nu$

```
Decay anti-B0
1.000 D*+ e- anti-nu_e     ISGW;
Enddecay
```

Notes:
This model does not include the A_3 form factor that is needed for non-zero mass leptons, i.e., tau's. If tau's are generated the A_3 form factor will be zero.

A.13 ISGW2

Author: Lange, Ryd
Usage:
`BrFr D1 D2 D3 ISGW2 ;`

Explanation:
This is a model for semileptonic decays of B, D, and D_s mesons according to the ISGW2 model [2]. The first daughter is the meson produced in the semileptonic decay. The second and third argument is the lepton and the neutrino respectively. See Section 12 for more details about semileptonic decays.

Example:

The example shows how to generate $\bar{B}^0 \to D^{*+} e\nu$

```
Decay anti-B0
1.000 D*+ e- anti-nu_e     ISGW2;
Enddecay
```

Notes:
This model has been fairly well tested for B decays, most form factors and distributions have been compared to the original code that we obtained from D. Scora.

A.14 JETSET

Author: Ryd, Waldi
Usage:
`BrFr D1 D2 DN JETSET MODE;`

Explanation:
A particle who's decay is not implanted in EvtGen can be decayed by calling JetSet using this model as an interface. The decays that uses the `JETSET` model are converted into the JetSet

decay table format and read in by JetSet. However, if JetSet produces a final state which is explicitly listed as another decay of the parent it is rejected. E.g. consider this example:

```
Decay J/psi
0.0602  e+  e-                        VLL;
0.0602  mu+ mu-                       VLL;
.
.
.
0.8430  rndmflav  anti-rndmflav       JETSET        12;
Enddecay
```

In this example if JetSet decays the J/Ψ to e^+e^- or $\mu^+\mu^-$ the decay is rejected and regenerated. For more details about the EvtGen-Jetset interface see Appendix E.

A.15 JSCONT

Author: Ryd, Kim
Usage:
`BrFr  JSCONT Flavor;`

Explanation:
This decay model is for generation of continuum events at the $\Upsilon(4S)$. It uses JetSet to fragment quark strings. The flavor of the primary string is given as the argument to the model and 1 means a $d\bar{d}$, 2 is $u\bar{u}$, 3 is $s\bar{s}$ and 4 is $c\bar{c}$. If the flavor is 0 a mixture of the quarks will be generated in the appropriate amounts. The first particle that is created is the VPHO which can be decayed using this decay model. The primary jets are created according to a $1+\cos^2\theta$ distribution, where θ is the angle of the primary jet with respect to the beam line, or more precisely the z-axis.

Example:

```
Decay VPHO
1.000          JSCONT 1;
Enddecay
```

A.16 KLL3P

Author: Rotondo
Usage:
`BrFr K L1 L2 KLL3P ;`

Explanation:
Implementation for the process $B \to Kl^+l^-$, as the previous model the form-factors are calculated in the framework of three-point QCD sum-rules [10].

Example:
The example shows how to generate $B^+ \to K^+e^+e^-$

```
Decay B+
1.000 K+ e+ e- KLL3P;
Enddecay
```

Notes:
Only Short Distance interaction are considered.

A.17 KSLLLCQCD

Author: Rotondo
Usage:
```
BrFr K* L1 L2 KSLLLCQCD ;
```

Explanation:
Implementation of the process $B \to K^* l^+ l^-$. In this model the hadronic part of the matrix element is calculated in the framework of the light-cone QCD sum rules [11].

Example:
The example shows how to generate $B^0 \to K^{*0}\tau^+\tau^-$.

```
Decay B0
1.000 K*0 tau+ tau- KSLLLCQCD;
Enddecay
```

Notes:
In the Aliev's paper [11] are taken in account some effects out the Standard-Model, this implementation recover only the SM part of the interaction.
Only Short Distance contribution are considered.
Warning: a cut-off on the q^2 of the lepton pair are introduced. The q^2 is required to be greater than $0.08 GeV^2$, to avoid a large spend of time for the generation of the right configuration. This approximation is not useful for $B \to K^* e^+ e^-$.

A.18 KSLL3PQCD

Author: Rotondo
Usage:
```
BrFr K* L1 L2 KSLL3PQCD ;
```

Explanation:
Implementation for the process $B \to K^* l^+ l^-$ in which the hadronic part of the matrix element is calculated in the framework of the three-point QCD sum rules [10].

Example:
The example shows how to generate $B^0 \to K^{*0}\mu^+\mu^-$.

```
Decay B0
1.000 K*0 mu+ mu- KSLL3PQCD;
Enddecay
```

Notes:
Only Short Distance interaction are considered.
Warning: as the previous model.

A.19 MELIKHOV

Author: Lange
Usage:
`BrFr M L NU MELIKHOV ;`

Explanation:
Implements the form factor model for $B \to \rho\ell\nu$ according to Melikhov, as described in hep-ph/9603340. There is one argument, which should be an integer between 1 and 4. The arguement sets which set of form factors from Melikhov should be used.

Example:
The example shows how to generate $B^0 \to \rho^- \mu^+ \nu_\mu$.

```
Decay B0
1.000 rho- mu+ nu_mu MELIKHOV 1;
Enddecay
```

Notes:

A.20 OMEGA_DALITZ

Author: Lange
Usage:
`BrFr P1 P2 P3 OMEGA_DALITZ ;`

Explanation:
The dalitz amplitude for the decay $\omega \to \pi^+\pi^-\pi^0$. The amplitude for this process is given by $A = \epsilon_{\mu\nu\alpha\beta} p^\mu_{\pi^+} p^\nu_{\pi^-} p^\alpha_{\pi^0} \varepsilon^\beta$.

Example:

```
Decay omega
1.000 pi+  pi-  pi0           OMEGA_DALITZ;
Enddecay
```

A.21 PHIGAMMA

Author: Zallo,Ryd
Usage:
`BrFr PHI GAMMA PHIGAMMA CSFWMN CSBKMN;`

Explanation:
Generates the interaction, $e^+e^- \to \phi\gamma$ according to [12]. This model should be used as a decay of the `VPHO`.

Example:
Example below shows how to generate the $\phi\gamma$ final state from an virtual photon.

```
Decay VPHO
1.000 phi gamma                    PHIGAMMA 0.878 0.95;
Enddecay
```

Notes:
This model produces an unpolarized ϕ.

A.22 PHSP

Author: Ryd
Usage:
`BrFr P1 P2 ...  PN PHSP ;`

Explanation:
Generic phase space to n-bodies. All spins of particles in the initial state and the final state are averaged.

Example:
As an example of using this model the decay $D^0 \to K^{*-}\pi^+\pi^0\pi^0$ is used.

```
Decay D0
1.000 K*-  pi+ pi0 pi0                    PHSP;
Enddecay
```

A.23 SINGLE

Author: Ryd
Usage:
`BrFr P SINGLE pmin pmax cthetamin cthetamax phimin phimax;`

Explanation:
Generates single particle rays in the region of phase space specified by the arguments. This single particle generator generates decays uniformly in the parents rest frame in the momentum range from `pmin` to `pmax`. However, the range of θ is specified in the lab frame.

Example:

Generates μ^+ with momentum from 0.5 to 1.0 GeV over 4π.

```
Decay Upsilon(4S)
1.000 mu+       SINGLE  0.5 1.0 -1.0 1.0 0.0 6.283185;
Enddecay
```

A.24 SLN

Author: Songhoon,Ryd
Usage:
`BrFr L N SLN ;`

Explanation:
This decay generates the decay of a scalar to a lepton and a neutrino. The amplitude for this process is given by $A = P^{\nu}\langle\ell|(V-A)_{\nu}|\nu\rangle$.

Example:

As an example of using this model the decay $D_s^+ \to \mu^+\bar{\nu}$ is used.

```
Decay DS+
1.000 mu+  anti-nu_mu                              SLN;
Enddecay
```

A.25 SLPOLE

Author: Lange
Usage:
`BrFr M L NU SLPOLE arguments;`

Explanation:
Implements a semileptonic decay according to a pole form parametrization. For definition of the form factors that are used see section 12.5.1.

Example:

The example shows how to generate $B^0 \to \rho^-\mu^+\nu_\mu$.

```
Decay B0
1.000 rho- mu+ nu_mu SLPOLE 0.27 -0.11 -0.75 1.0 0.23 -0.77
             -0.40 1.0 0.34 -1.32 0.19 1.0 0.37 -1.42 0.50 1.0;

Enddecay
```

Notes:

A.26 SSS_CP

Author: Ryd
Usage:
`BrFr S S SSS_CP ALPHA dm cp |A| argA |barA| argbarA;`

Explanation:
Decay of a scalar to two scalar and allows for CP violating time asymmetries. The first argument is the relevant CKM angle in radians. The second argument is the mass difference in s$^-1$ (approx 0.5×10^{12}). cp is the CP of the final state, it is ± 1. Next is the amplitude of a B^0 to decay to the final state, where the third argument is the magnitude of the amplitude and the fourth is the phase. The last two arguments are the magnitude and phase of the amplitude for a decay of a $\bar{B}^0$ to decay to the final state. This model then uses these amplitudes together with the time evolution of the $B\bar{B}$ system and the flavor of the other B to generate the time distributions.

Example:

This example decays the B meson to $\pi+\pi^-$

```
Decay B
1.000 pi+  pi-                    SSS_CP alpha dm 1.0 0.0 1.0 0.0;
Enddecay
```

Notes:
For more details about the treatment of CP violating decays see Section 11.

A.27 SSS_CP_PNG

Author: Ryd, Kuznetsova
Usage:
`BrFr S S SSS_CP_PNG BETA GAMMA DELTA dm cp` $|A_{tree}|$ $|A_{tree}|/|A_{penguin}|$;

Explanation:
This model takes into account penguin contributions in $B \to \pi\pi$ decays. It assumes single (top) quark dominance for the penguin. The first two arguments are the relevant CKM angles in radians; the third argument is the relative strong phase in radians; dm is the mass difference in s^{-1} (approx 0.5×10^{12}); cp is the CP of the final state; $|A_{tree}|$ is the tree-level amplitude, and $|A_{tree}|/|A_{penguin}|$ is the ratio of the amplitudes for the tree and penguin diagrams (≈ 0.2 for this decay mode). This model automatically takes into account the correct number of B^0 tags for this decay, which is given by:

$$f = \frac{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right)}{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right) + |A_f|^2 \left(1 + |r_f|^2 + \frac{(1-|r_f|^2)}{1+x_d^2}\right)} \tag{60}$$

where $x_d \equiv \frac{\Delta m}{\Gamma} \approx 0.65$, and

$$r_f = e^{2i\,\phi_M} \frac{\overline{A}_f}{A_f},\ \overline{r}_f = \frac{1}{r_f} \tag{61}$$

ϕ_M being the mixing angle, and the amplitude A_f being:

$$A_f \equiv A(B^0 \to \pi^+\pi^-) = A_t\, e^{i\phi_t} + A_p\, e^{i\phi_p}\, e^{i\delta} \tag{62}$$

with

$$\overline{A}_f \equiv A(\overline{B}^0 \to \pi^+\pi^-) = A_t\, e^{-i\phi_t} + A_p\, e^{-i\phi_p}\, e^{i\delta} \tag{63}$$

Here, A_t, ϕ_t are tree-level amplitude and phase, respectively, A_p, ϕ_p are those for the penguin, and δ is the relative strong phase.

Example:

This example generates $B^0 \to \pi^+\pi^-$.

```
Decay B0
1.000 pi+  pi-    SSS_CP_PNG beta gamma 0.1 dm 1.0 1.0 0.2;
Enddecay
```

Notes:
For more details about the treatment of CP violating decays see section 11.

A.28 STS

Author: Ryd
Usage:
`BrFr T S STS ;`

Explanation:
This model decays a scalar meson to a tensor and a scalar.

Example:

This example decays the B^+ meson to $D_2^*0\pi^+$

```
Decay B+
1.000 D_2*0  pi+                    STS;
Enddecay
```

Notes:

A.29 STS_CP

Author: Ryd
Usage:
`BrFr T S STS_CP ALPHA dm cp |A| argA |barA| argbarA;`

Explanation:
Decay of a scalar to a tensor and a scalar and allows for CP violating time asymmetries. The

first argument is the relevant CKM angle in radians. The second argument is the mass difference in s$^-1$ (approx 0.5×10^{12}). cp is the CP of the final state, it is ± 1. Next is the amplitude of a B^0 to decay to the final state, where the third argument is the magnitude of the amplitude and the fourth is the phase. The last two arguments are the magnitude and phase of the amplitude for a decay of a $\bar{B}^0$ to decay to the final state. This model then uses these amplitudes together with the time evolution of the $B\bar{B}$ system and the flavor of the other B to generate the time distributions.

Example:

This example decays the B meson to $a_2^0\pi^0$

```
Decay B0
1.000 a_20  pi0                       STS_CP alpha dm 1.0 0.0 1.0 0.0;
Enddecay
```

Notes:
For more details about the treatment of CP violating decays see section 11.

A.30 SVP_HELAMP

Author: Ryd
Usage:
`BrFr V P SVP_HELAMP |H+| argH+ |H-| argH-;`

Explanation:
The decay of a scalar to a vector and a photon. This decay is parameterized by the helicity amplitudes H_+ and H_-. For more information about helicity amplitudes see Section F/

Example:

```
Decay B0
1.000 K*0  gamma                       SVP_HELAMP 1.0 0.0 1.0 0.0;
Enddecay
```

A.31 SVS

Author: Ryd
Usage:
`BrFr V S SVS ;`

Explanation:
The decay of a scalar to a vector and a scalar. The first daughter is the vector meson.

Example:

As an example we consider $B \to D^*\pi$.

```
Decay B0
1.000 D*+  pi-                SVS;
Enddecay
```

A.32 SVS_CP

Author: Ryd
Usage:
`BrFr V S SVS_CP ALPHA dm cp |A| argA |barA| argbarA;`

Explanation:
Decay of a scalar to a vector and a scalar and allows for CP violating time asymmetries. The first daughter has to be the vector. The first argument is the relevant CKM angle in radians. The second argument is the mass difference in s$^-$1 (approx 0.5×10^{12}). cp is the CP of the final state, it is ± 1. Next is the amplitude of a B^0 to decay to the final state, where the third argument is the magnitude of the amplitude and the fourth is the phase. The last two arguments are the magnitude and phase of the amplitude for a decay of a $\bar{B}^0$ to decay to the final state. This model then uses these amplitudes together with the time evolution of the $B\bar{B}$ system and the flavor of the other B to generate the time distributions.

Example:
This example decays the B^0 meson to $J/\Psi K_s$

```
Decay B0
1.000 J/psi  K_S0                 SVS_CP beta dm 1.0 0.0 1.0 0.0;
Enddecay
```

Notes:
For more details about the treatment of CP violating decays see Section 11.

A.33 SVS_CP_ISO

Author:NK
Usage:
BrFr V S SVS_CP_ISO beta dm flip $|T^{+0}|$ argT^{+0} $|\overline{T^{+0}}|$ arg$\overline{T^{+0}}$ $|T^{0+}|$ argT^{0+} $|\overline{T^{0+}}|$ arg$\overline{T^{0+}}$ $|T^{+-}|$ argT^{+-} $|\overline{T^{+-}}|$ arg$\overline{T^{+-}}$ $|T^{-+}|$ argT^{-+} $|\overline{T^{-+}}|$ arg$\overline{T^{-+}}$ $|P_0|$ argP_0 $|\overline{P_0}|$ arg$\overline{P_0}$ $|P_2|$ argP_2 $|\overline{P_2}|$ arg$\overline{P_2}$;

Explanation:
This model considers B decays into a vector (V) and a scalar (S) from the point of view of isospin analysis. The vector should always be listed first. For the three B^0 (or $\overline{B}^0$) modes ($B^0 \to V^+S^-$, $B^0 \to V^-S^+$, and $B^0 \to V^0S^0$), it takes into account mixing, and generates the corresponding CP-violating asymmetries. It can also be used for the two isospin-related B^+ (B^-) modes (e.g., $B^+ \to V^+S^0$ and $B^+ \to V^0S^+$), as all five modes should be treated together in this approach. Following the conventions of Lipkin, Nir, Quinn, and Snyder (Phys. Rev. D**44**, 1454 (1991)),

the various decay amplitudes can be written as follows:

$$A(B^+ \to V^+ S^0) \equiv \sqrt{2} A^{+0} = T^{+0} + 2P_1 \tag{64}$$

$$A(B^+ \to V^0 S^+) \equiv \sqrt{2} A^{0+} = T^{0+} - 2P_1 \tag{65}$$

$$A(B^0 \to V^+ S^-) \equiv A^{+-} = T^{+-} + P_1 + P_0 \tag{66}$$

$$A(B^0 \to V^- S^+) \equiv A^{-+} = T^{-+} - P_1 + P_0 \tag{67}$$

$$A(B^0 \to V^0 S^0) \equiv 2A^{00} = T^{0+} + T^{+0} - T^{-+} - T^{+-} - 2P_0 \tag{68}$$

where the amplitudes T^{ij} contain no penguin contributions, P_1 is penguin amplitude for the final $I = 1$ state, and P_0, for the final $I = 0$ state.

The model's arguments are:

- beta = corresponding CKM angle
- dm = $B^0\overline{B}^0$ mass difference ($\approx 0.5 \times 10^{12} s^{-1}$).
- "flip" sets the fraction of $B \to f$ to $B \to \overline{f}$ decays, where the state specified in the .DEC table is considered the "f" state. Set it to 0 to always get the $B \to f$ case, and to 1 to always get the $B \to \overline{f}$ case.
- $|T^{+0}|$, $\varphi_{T^{+0}}$ = magnitude and phase of the corresponding amplitude
- $|\overline{T^{+0}}|$, $\varphi_{\overline{T^{+0}}}$ = magnitude and phase of the corresponding amplitude for the CP-conjugate process.
- $|T^{0+}|$, $\varphi_{T^{0+}}$ = magnitude and phase of the corresponding amplitude
- $|\overline{T^{0+}}|$, $\varphi_{\overline{T^{0+}}}$ = magnitude and phase of the corresponding amplitude for the CP-conjugate process.
- $|T^{+-}|$, $\varphi_{T^{+-}}$ = magnitude and phase of the corresponding amplitude
- $|\overline{T^{+-}}|$, $\varphi_{\overline{T^{+-}}}$ = magnitude and phase of the corresponding amplitude for the CP-conjugate process.
- $|T^{-+}|$, $\varphi_{T^{-+}}$ = magnitude and phase of the corresponding amplitude
- $|\overline{T^{-+}}|$, $\varphi_{\overline{T^{-+}}}$ = magnitude and phase of the corresponding amplitude for the CP-conjugate process.
- $|P_0|$, φ_{P_0} = magnitude and phase of the corresponding amplitude
- $|\overline{P_0}|$, $\varphi_{\overline{P_0}}$ = magnitude and phase of the corresponding amplitude for the CP-conjugate process.
- $|P_2|$, φ_{P_2} = magnitude and phase of the corresponding amplitude
- $|\overline{P_2}|$, $\varphi_{\overline{P_2}}$ = magnitude and phase of the corresponding amplitude for the CP-conjugate process.

Example:

This example decays the B^0 meson to $a_1^- \pi^+$ assuming no penguin contributions

```
Decay B0
1.000 a_1-  pi+      SVS_CP_ISO beta dm 0.0 1.0 0.0 1.0 0.0
                                               1.0 0.0 1.0 0.0
                                               1.0 gamma 3.0 -gamma
                                               3.0 gamma 1.0 -gamma
                                               0.0  0.0  0.0 0.0;

Enddecay
```

Notes:
For more details about the treatment of CP violating decays see section 11.

A.34 SVS_NONCPEIGEN

Author: Natalia,Ryd
Usage:
BrFr V S SVS_NONCPEIGEN alpha dm flip $|A_f|$ argA_f $|\overline{A}_f|$ arg$\overline{A}_f$ $|A_{\overline{f}}|$ arg$A_{\overline{f}}$ $|\overline{A}_{\overline{f}}|$ args$\overline{A}_{\overline{f}}$;

Explanation:
This model allows to generate scalar $\rightarrow$ vector + scalar decays, where the final state is not a CP-eigenstate. The `flip` parameter sets the fraction of f to $\bar{f}$ decays, where the state specified in the .DEC table is considered the "f" state. Set it to 0 to always get the final f case, and to 1 to always get the $\overline{f}$ final state. This model automatically generates the correct number of B^0 and $\bar{B}^0$ tags, depending on the specified amplitudes.

Example:

This example will generate a mixture of $a_1^+ \pi^-$ and $a_1^- \pi^+$ final states with the appropriate number of B^0 and $\bar{B}^0$ tags.

```
Alias MYB B0
Decay Upsilon(4S)
1.00  MYB B0
Enddecay
Decay MYB
1.000 a_1- pi+    SVS_NONCPEIGEN alpha dm 0.5
                                           1.0 0.0 3.0 0.0
                                           3.0 0.0 1.0 0.0;
Enddecay
```

Note that 0.5 will 50% of the time charge conjugate the final state $a_1^- \pi^+$ and that the tag B, the other B in the $\Upsilon(4S)$ decay will be selected as either a B^0 or a $\bar{B}^0$ according to the amplitudes that are specified. Also, note that the B that decays into the CP mode has to be an alias for either a B^0 or a $\bar{B}^0$, it can not be a B^0 or a $\bar{B}^0$ directly, since the SVS_NONCPEIGEN model will create the other B either as a B^0 or a $\bar{B}^0$!

Notes:
For more details about the treatment of CP violating decays see Section 11.

A.35 SVV_CP

Author: Ryd
Usage:
```
BrFr V1 V2 SVV_CP BETA dm eta |G1+| argG1+ |G0+| argG0+
|G1-| argG1-;
```

Explanation:
Decay of a scalar to two vector mesons and allows for CP violating time asymmetries. The first argument is the relevant CKM angle in radians. The second argument is the $B^0 - \bar{B}^0$ mass difference in s^{-1} (approximately 0.5×10^{12}). The next argument is called η in Ref. [13] and is either $+1$ or -1. The last six arguments are G_{1+}, G_{0+}, and G_{1-}, and are expressed as their absolute values and phases again the definition of these parameters are in Ref. [13]. This model then uses these amplitudes together with the time evolution of the $B\bar{B}$ system and the flavor of the other B to generate the time distributions.

Example:
This example decays the B^0 meson to $J/\Psi K^{*0}$

```
Decay B0
1.000 J/psi  K*0        SVV_CP beta dm 1.0 1.0 0.0 1.0 0.0 1.0 0.0;
Enddecay
```

Notes:
For more details about the treatment of CP violating decays see Section 11. Note that the value of η depends on how the K^{*0} decays, it is either $+1$ or -1 depending on weather a K_S or a K_L is produced. (It needs to be checked which sign goes with the K_S and the K_L.

A.36 SVV_HELAMP

Author: Ryd
Usage:
```
BrFr V1 V2 SVV_HELAMP |H+| argH+ |H0| argH0 |H-| argH-;
```

Explanation:
The decay of a scalar to two vectors. The decay amplitude is specified by the helicity amplitudes which are given as arguments for the decay. The arguments are H_+, H_0, and H_-. Where these complex amplitudes are specified as magnitude and phase. The convention for the helicity amplitudes are that of Jacob and Wick (at least I hope this is what it is!). For more details about helicity amplitudes see Section F.

Example:

```
Decay D0
1.000 K*0 rho0           SVV_HELAMP 1.0 0.0 1.0 0.0 1.0 0.0;
Enddecay
```

A.37 TAULNUNU

Author: Ryd
Usage:
`BrFr L N1 N2 TAULNUNU ;`

Explanation:
The decay of a tau to a lepton and two neutrinos. The first daughter is the produced lepton the second is the associated neutrino and the third is the tau neutrino. The amplitude for this decay is given by $A = \langle\tau|(V-A)_\alpha|\nu_\tau\rangle\langle\ell|(V-A)^\alpha|\nu_\ell\rangle$.

Example:
The example shows the decay $\tau \to e\nu_e\bar{\nu}_t au$

```
Decay tau-
1.000 e- anti-nu_e  nu_tau              TAULNUNU;
Enddecay
```

A.38 TAUSCALARNU

Author: Ryd
Usage:
`BrFr S N TAUSCALARNU`

Explanation:
The decay of a tau to a scalar meson and a tau neutrino. The meson is the first daughter. The amplitude for this decay is given by $A = \langle\tau|(V-A)_\alpha|\nu_\tau\rangle P^\alpha$.

Example:
An example of the use of this model is in the decay $\tau \to \pi\nu_\tau$

```
Decay tau-
1.000 pi-   nu_tau              TAUSCALARNU;
Enddecay
```

A.39 TAUVECTORNU

Author: Ryd
Usage:
`BrFr V N TAUVECTORNU`

Explanation:
The decay of a tau to a vector meson and a tau neutrino. The meson is the first daughter. The amplitude for this decay is given by $A = \langle\tau|(V-A)_\alpha|\nu_\tau\rangle\varepsilon^\alpha$.

Example:

An example of the use of this model is in the decay $\tau \to \rho\nu_\tau$

```
Decay tau-
1.000 rho-   nu_tau              TAUVECTORNU;
Enddecay
```

A.40 TSS

Author: Ryd
Usage:
`BrFr S1 S2 TSS`

Explanation:
The decay of a tensor particle to two scalar mesons.

Example:

As an example the decay $D_2^{*0} \to D^0\pi^0$is used.

```
Decay D_2*0
1.000 D0   pi0              TSS;
Enddecay
```

A.41 TVS_PWAVE

Author: Ryd
Usage:
`BrFr V S TVS_PWAVE |P| argP |D| argD |F| argF;`

Explanation:
The decay of a tensor particle to a vector and a scalar. The decay takes six arguments, which parameterizes the P, D, and F wave amplitudes. The first two arguments are the magnitude and the phase of the P-wave amplitude, the third and forth are the D-wave amplitude and the last two are the F-wave amplitude.

Example:

The decay $D_2^{*0} \to D^{*0}\pi^0$ which is expected, by HQET, to be dominated by D wave.

```
Decay D_2*0
1.000 D*0 pi0           TVS_PWAVE 0.0 0.0 1.0 0.0 0.0 0.0;
Enddecay
```

Notes:
This model has only been used yet for D-wave so further test are needed before it is safe to use for nonzero P and F wave amplitudes.

A.42 VLL

Author: Ryd
Usage:
`BrFr L1 L2 VLL ;`

Explanation:
Decay of a vector meson to a pair of charged leptons, e.g., $J/\psi \to \ell^+\ell^-$. The amplitude for this process is given by $A = \varepsilon^\mu L_\mu$ where $L_\mu = \langle \ell | V_\mu | \bar{\ell} \rangle$.

Example:
The example shows how to generate $J/\Psi \to e^- e^+$

```
Decay J/psi
1.000 e- e+     VLL;
Enddecay
```

A.43 VSP_PWAVE

Author: Ryd
Usage:
`BrFr S gamma VSP_PWAVE ;`

Explanation:
The decay of a vector to a scalar meson and a photon, the decay goes in P-wave. The first daughter is the scalar meson and the second daughter is the photon.

Example:
This decay is useful for example in the decay $D^{*0} \to D^0\gamma$

```
Decay D*0
1.000 D0  gamma                    VSP_PWAVE;
Enddecay
```

A.44 VSS

Author: Ryd
Usage:
`BrFr S1 S2 VSS ;`

Explanation:
Decays a vector particle into two scalars. It generates the correct decay angle distributions for the

produced scalars. The amplitude for this decay is given by $A = \varepsilon^\mu v_\mu$ where ε is the polarization vector of the parent particle and the v is the (four) velocity of the first daughter.

Example:

The example shows how to generate $D^{*+} \to D^0\pi^+$

```
Decay D*+
1.000 D0 pi+   VSS;
Enddecay
```

A.45 VSS_MIX

Author: Ryd
Usage:
`BrFr B1 B2 VSS_MIX dm;`

Explanation:
Decays a vector particle into two scalar and generates the correct angular and time distributions for the particles in the decay $\Upsilon(4S) \to B^0\bar{B}^0$. The mass difference is supplied as an argument to the model

Example:

The example shows how to generate the mixture of mixed and unmixed B^0 and $\bar{B}^0$ events.

```
Define dm 0.474e12
Decay Upsilon(4S)
0.420 B0      anti-B0                                VSS_MIX dm;
0.040 anti-B0 anti-B0                                VSS_MIX dm;
0.040 B0      B0                                     VSS_MIX dm;
Enddecay
```

Notes:
The user has to manually specify the fractions of mixed and un-mixed event through the branching fraction. This means that all this model does is to generate the right time distribution for the given final state. See Section 11 for more details about how mixing is implemented and how it works with CP violation.

A.46 VVS_PWAVE

Author: Ryd
Usage:
`BrFr V S VVS_PWAVE |S| argS |P| argP |D| argD;`

Explanation:
The decay of a vector particle to a vector and a scalar. The decay takes six arguments, which parameterizes the S, P, and D wave amplitudes. The first two arguments are the magnitude

and the phase of the S-wave amplitude, the third and forth are the P-wave amplitude and the last two are the D-wave amplitude.

Example:

The example below shows how to decay the a_1^0 in pure P wave to $\rho\pi$.

```
Decay a_10
1.000 rho0 pi0          VVS_PWAVE 0.0 0.0 1.0 0.0 0.0 0.0;
Enddecay
```

Notes:
This model has only been used yet for P-wave so further test are needed before it is safe to use use for nonzero S and D wave amplitudes.

A.47 WSB

Author: Lange, Ryd
Usage:
`BrFr M L N WSB ;`

Explanation:
This is a model for semileptonic decays of B, and D mesons according to the WSB model [14]. The first daughter is the meson produced in the semileptonic decay. The second and third argument is the lepton and the neutrino respectively. See Section 12 for more details about semileptonic decays.

Example:

The example shows how to generate $\bar{B}^0 \to D^{*+} e\bar{\nu}$

```
Decay anti-B0
1.000 D*+ e- anti-nu_e     WSB;
Enddecay
```

Notes:
This model does not include the A_3 form factor that is needed for non-zero mass leptons, i.e., tau's. If tau's are generated the A_3 form factor will be zero.

B EvtGen interface

The EvtGen package supplies a test program, testEvtGen.cc, which provides a main routine that tests various functions of the EvtGen package. However, a user of the EvtGen package is not expected to use this program in his application. Instead it is expected that the event loop needed to generate the events is written externally and invokes routines in EvtGen to do initialization and event generation. This section describes the various functions that makes the user interface.

The interface is provided through the class `EvtGen` that provides member functions to do initialization of the generator, normally prior to generating events, and to generate events. It also provides access to controll of various configurations parameters that are used to control the generator.

Initialization of the generator is done with the function

```
void EvtGen::init(char* decay, char* pdt, char *udecay);
```

where `decay` is the name of the decay table to use, `pdt` is the name of the particle property file that is used by EvtGen and `udecay` is the optional name of a user decay table that is read after the main decay table, `decay`. If `udecay` is either an empty string or an NULL pointer then no user decay file is read. If you want to read additional decay files this can be done with the

```
EvtDecayTable::ReadDecay(char * decay);
```

member function.

EvtGen can after this initialization be asked to generate one event at the time by calling the member function

```
void EvtGen::event(int stdhepid,LorentzVector p,LorentzVector d);
```

where `stdhepid` is the particle number according to the stdhep particle numbering scheme. `p` is a four-vector giving the initial momentum and energy of the initial particle. Similarly. `d` gives the initial vertex and time for the initial particle. This function will generate an event and then store it in the stdhep comon block, `hepevt`. This is the most convenient form for having EvtGen to generate an event. In some test that are internal to EvtGen it is not an advantage to have EvtGen fill the stdhep common block. Instead these test work with the internal EvtGen structures. Here the user will need to create the initial particle and then generate the decay with the memebr function

```
void EvtGen::event(EvtParticle* p);
```

Note that here the user is also responsible for destroying the the created decay structure. This is done by

```
p->deleteTree();
```

For the convienence of users more familar with fortran, we have also provided a fortran frontend to the `EvtGen` interface described above. There are three necesary files:

- `top.cc`
- `myfunc.F`
- `evtgenevent_.cc`

The first of these simply is the main that calls `myfort.F`. In order to link with a C++ compiler, there must be a main writen in C++. `evtgeninit_` and `evtgenevent_` are the routines that actually call the `EvtGen:Init` and `EvtGen:Decay` routines. These routine handles the initialization of EvtGen as well as the deletion of the particle tree from memory when it is no longer needed. `evtgenint_` takes three arguements, and is declared as

```
void evtgeninit_(char *decayname, char *udecayname, char *pdttablename)
```

`evtgenevent_` also takes three input arguments:

```
void evtgenevent_(float *pos, float *tim, int *parent)
```

`pos` is an array that contains the initial x, y, and z position of the particle. `tim` is the production time of the particle. `parent` is the standard HEP number of the particle. The EvtGen output is returned in the standard HEP common block defined in `stdhep/stdhep.inc` `myfort.F` is the fortran routine that should be modified to suit your needs. By default it will generate one event with a B^0 meson as the parent.

It is straightforward to modify this interface structure in order to include EvtGen inside of a larger fortran program. `top.cc` should be modified to call whichever routine is currently the top fortran routine. The routine `myfort.F` can then be modified to suit your needs.

C Final state radiation

To incorporate the effects of final state radiation an interface to PHOTOS [15] has been added. Photos is a packaged that generate final state photons on a generated decay. In the case of the decay $J/\Psi \to e^+e^-$ a large fraction of the electrons and positrons in the final state will emit one (or more) photons.

PHOTOS is activated in the decay file for a particular channel by adding "PHOTOS" before the name of the decay model,

```
Decay J/psi
1.0000  e+ e-       PHOTOS VLL;
Enddecay
```

PHOTOS is useful for radiating photons of any charged particles in the final state, maybe someone should try it on $B \to \pi^+\pi^-$!

D Performance monitoring

This section describes various tools that are available to monitor and debug the performance of EvtGen.

D.1 Efficiency

One, of many, possible reason for the generator to run slowly is that the maximum probability, against which the acceptance-rejection method is applied, is to high. This means that the efficiency for accepting an event is low. Diagnostics of this can be studied by setting the environment variable "EVTINFO". At the end of the job a summary will then be printed out which gives the number of times each decay model was invoked, the efficiency for accepting the decay and the ratio of the largest probability in the sample that was generated to the maximum probability. Optimal running condition wants to maximize both of these.

E Interface to JetSet

The `JETSET` and `JSCONT` models provides interface to JetSet. The `JSCONT` model is specialized for producing $q\bar{q}$ jets from an e^+e^- interaction. The `JETSET` model is a generic interface to use JetSet for decaying particles.

When JetSet is asked to decay a particle it uses a decay table that has been build from the EvtGen decay table, DECAY.DEC. Entries in the decay table that uses the `JETSET` model is copied into a JetSet format decay table. This decay table is read by JetSet after EvtGen is done reading the decay tables, including user decay tables. (This file is currently called jet.d and is left after the jobs is finished in the current directory.) The decay table for JetSet is constructed from the list of daughters that were listed in the EvtGen decay table and the model is taken as the argument to the `JETSET` model.

To implement the partons that jetset generates a new particle in EvtGen was created to hold the list of partons. This class is called EvtStringParticle, the name is taken from the idea in JetSet that these partons form a string. The EvtStringParticle is derived from EvtParticle, and addes member data to store the four momenta and ids of the partons.

F Details about helicity amplitudes

Warning! This section is still not complete, and could possibly contain serios errors. But if you read it and have any comments questions, please send them to ryd@stanford.slac.edu.

This section will deal with some details related to helicity ampltudes and their relations to partial wave amplitudes. In particular the the sign conventions are described and the relations between the Jackson and the Jackob-Wick conventions for the helicity formalism is explained.

At the moment this is not meant as a comlete guide to the use of helicity amplitudes. There are several references that gives a good intruduction to the use of helicity amplitudes for describing the dynamics of particle decays. Richman [16] gives a pedagodical introduction following the conventions of Jacob and Wick [17]. Jackson [18] uses a slightly different choice of conventions. The purpose of this section is to describe these conventions and establish a map between the two conventions. Both of these conventions are in use in the literature and, unfortunately, it is not always clear which convention is used.

The origin of the choice of conventions comes from how the Euler angles are chosen.

F.1 Preliminaries and definitions.

A few definitions are stated explicitly here as to avoid confusion about what conventions are used.

First, the basis vectors $\hat{x}$, $\hat{y}$, and $\hat{z}$ are said to form a right handed coordinate system if rotating the $\hat{x}$-axis 90 degrees counter-clockwise, as seen from the positive $\hat{z}$ directions, will take it to the direction of the $\hat{y}$ directions. It is assumed that $\hat{x}$, $\hat{y}$, and $\hat{z}$ are mutually orthogonal. If the coordinate system is not righthanded it is left handed – there is no other alternative. In the discussion below all coordinates considered are righthanded.

The Euler angles, α, β, and γ defines a rotation $R(\alpha,\beta,\gamma)$ by

$$R(\alpha,\beta,\gamma) = R_{z'}(\gamma)R_{y'}(\beta)R_z(\alpha) \tag{69}$$

where this means that a rotation by α is first performed around the z-axis. Then a rotation by β around the z'-axis is done. Where now the z'-axis is the new axis as obtained after the first rotation. Last the rotation around the new z-axis, z' is performed by an amount given by γ. Note that these rotations are not according to a fixed set of rotation axis. This is inconvenient as the rotation operators that we have are with respect to a fixed coordinate system. However, there is a simple way of rewriting the Euler rotation interms of rotations around a fixed coordinate system,

$$R(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_z(\gamma). \tag{70}$$

See e.g. Sakurai [**?**] page xxx for illustrations of the rotations.

F.2 Helicity amplitudes.

Consider the decay $A \to BC$. The particle A is described by the state $|J, m_z = \lambda_A\rangle >$ in the coordinate system x, y, and z. Conventionally, the z axis has been choosen as the quantization axis.

The particle B, produceded in the decay, will have a direction which will be labeled $\hat{p}$. The polar coordinates for this directions, as measured in A's coordinate system will be lables θ_B and ϕ_B. To write down the amplitude for this decay a choice of Euler angles has to be made that rotates the original z axis to the directions $\hat{p}$. This can be acomplished by chosing $\alpha = \phi_B$ and $\beta = \theta_B$. However, the azimuthal angle γ around the new z axis, z', is arbitrary. This is the source of conventions for the helicity formalism. The two common conventions are the Jackson conventions which takes $\gamma = 0$ and the Jackob-Wick conventions which takes $\gamma = -\phi_B$.

Given the choice of the Euler angels, α, β, and γ the amplitude for the decay can be written as

$$\begin{aligned} A_{\lambda_A\lambda_B\lambda_C} &= sqrt\frac{2J+1}{4\pi}H_{\lambda_B\lambda_C}D^{J*}_{\lambda_A,\lambda_B-\lambda_C}(\alpha\beta\gamma) && (71)\\ &= sqrt\frac{2J+1}{4\pi}H_{\lambda_B\lambda_C}e^{i\lambda_A\alpha+i(\lambda_B-\lambda_C)\gamma}d^J_{\lambda_A,\lambda_B-\lambda_C}(\beta) && (72)\\ & && (73) \end{aligned}$$

Using the explicit choices for the Euler angles according to the conventions we obtain the amplitudes according to the two different conventions

$$\begin{aligned} A^{Jack}_{\lambda_A\lambda_B\lambda_C} &= sqrt\frac{2J+1}{4\pi}H^{Jack}_{\lambda_B\lambda_C}e^{i\lambda_A\phi_A}d^J_{\lambda_A,\lambda_B-\lambda_C}(\beta) && (74)\\ A^{JW}_{\lambda_A\lambda_B\lambda_C} &= sqrt\frac{2J+1}{4\pi}H^{JW}_{\lambda_B\lambda_C}e^{i(\lambda_A-(\lambda_B-\lambda_C))\phi_A}d^J_{\lambda_A,\lambda_B-\lambda_C}(\beta) && (75)\\ & && (76) \end{aligned}$$

Since both of these amplitudes describe the same physics process it is tempting to equate the two amplitudes and determine the relation between $H^{Jack}_{\lambda_B\lambda_C}$ and $H^{JW}_{\lambda_B\lambda_C}$. But this in note correct since, they don't infact describe the same system! This is due to different phases that has been picked up by different rotations in the two conventions.

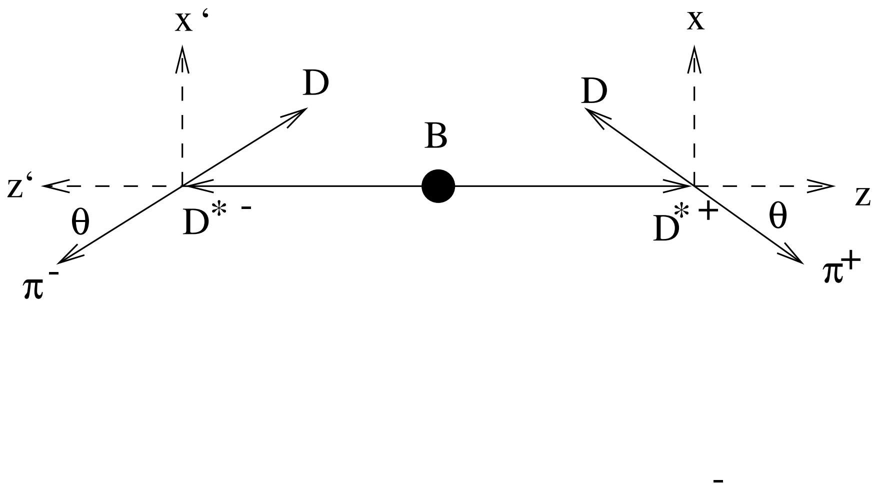

Figure 6: The coordinate systems in the Jackson convention. The two x axises are parallell.

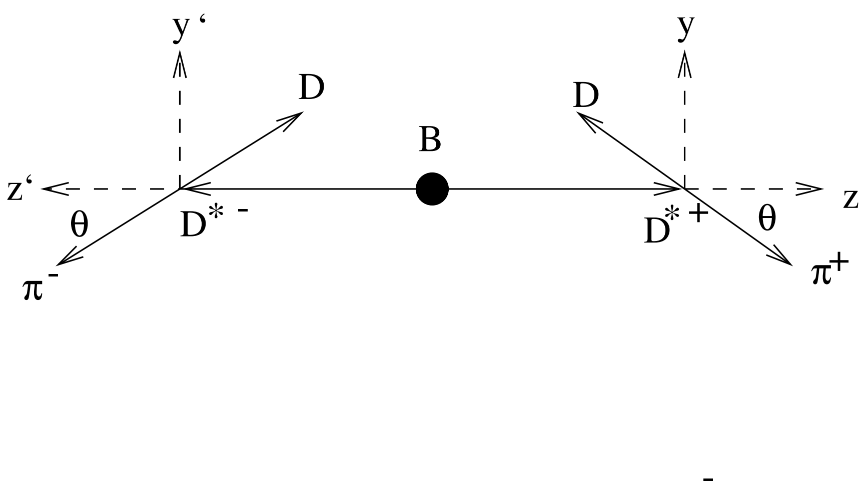

Figure 7: The coordinate systems in the Jacob-Wick convention. The two y axises are parallell.

F.3 Example.

The have a concrete example to discuss, the decay $B \to D^{*+}D^{*-}$ with $D^{*+} \to D^0\pi+$ and $D^{*-} \to \bar{D}^0\pi-$ is considered.

First we discuss how the coordinat systems are set up in the two different conventions. This is shown in figures 6 and 7. In both conventions the z axis of the coordinate system that describes the decay of the $D*$'s is along the direction of the flight of the D^*. But in the Jacob-Wick convention the direction of the y axises are parallell while in the Jackson convention the direction of the x axises are parallell.

G The routines `decay_angle` and `decay_angle_chi`

This section will describe two utility routines, `decay_angle` and `decay_angle_chi`, that compute decay angles. The function `decay_angle` is useful for finding the decay angle, as known as the helicity angle, in a decay tree that includes a two body decay. `decay_angle_chi` computes the azimuthal angle between two decay planes in sequencial decays such as $B \to D^* D^*$ or $B \to D^* \ell\nu$.

To call to `decay_angle`, the syntax is

```
costheta=decay_angle(P,Q,D),
```

where `P`, `Q` and `D` are of type `EvtVector4R`. This routine returns the cosine of the angle θ, as defined in Figure 8. The decay angle calculated is that between the flight direction of the daughter meson, `D`, in the rest frame of `Q` (the parent of `D`), with respect to `Q`'s flight direction in `P`'s (the parent of `Q`) rest frame. `P`, `Q`, and `D` are the momentum four vectors of these particles in any frame of reference. The decay angle is computed using the (manifestly invariant) expression

$$\cos\theta = \frac{(P \cdot D)M_Q^2 - (P \cdot Q)(Q \cdot D)}{\sqrt{[(P \cdot Q)^2 - M_Q^2 M_P^2][(Q \cdot D)^2 - M_Q^2 M_D^2]}}. \tag{77}$$

The routine `decay_angle_chi` is used to calculate the azimuthal angel, χ, between the decay planes of a pair of two body decays. As illustrated in Figure 9, χ is the angle calculated from the particle `d1` to the particle `h1` as seen from the direction of particle `D`. The form of the call to this subroutine is

```
chi=decay_angle_chi(p4_parent,p4_d1,p4_d2,p4_h1,p4_h2),
```

where `p4_d1`, `p4_d2`, `p4_t1`, `p4_h2` are of type `EvtVector4R` and are the four momenta of the daughter particles as illustrated in Figure 9.

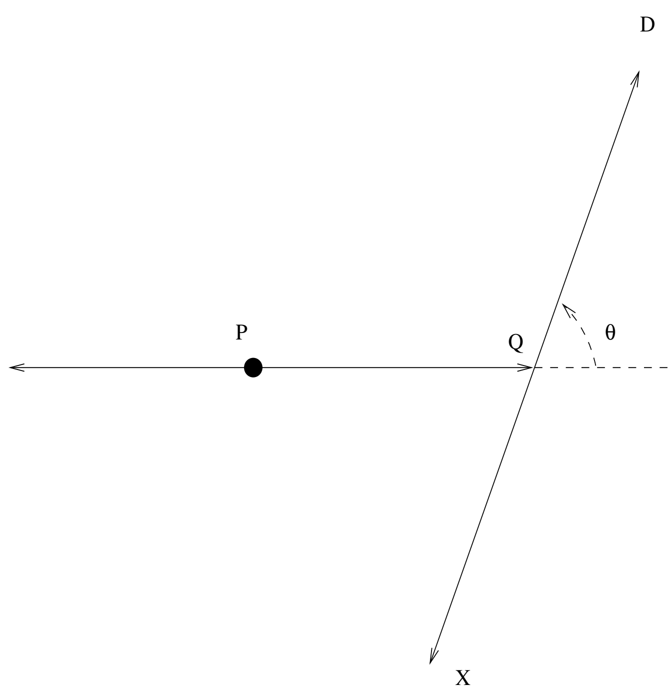

Figure 8: Definition of the decay angle.

H Optimizing EvtGen code

Broadly speaking there are two different categories of optimization. The firt type concerns just making the code more efficient, i.e., there are no changes to the algorithm. A very simple example

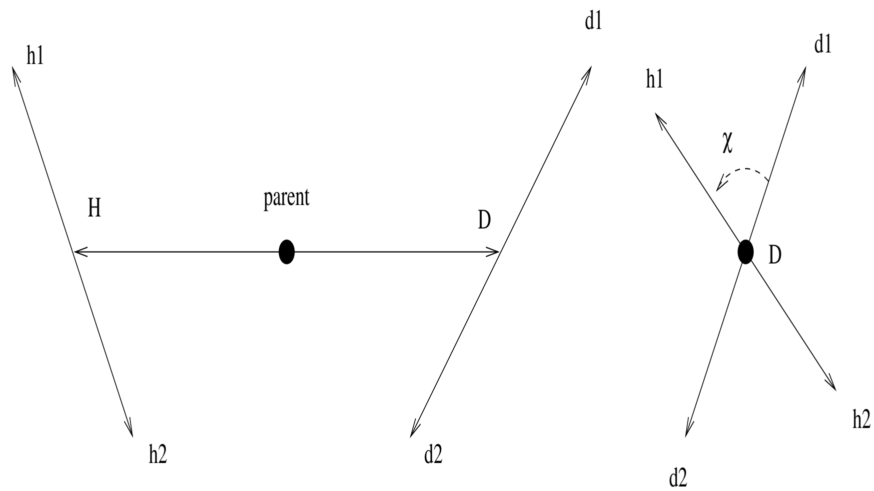

Figure 9: Definition of the χ angle.

of such optimization is the order in which operations are performed, let a be a double and p and k four-vectors.

```
a*(p*k);
```

is more efficient then

```
(a*p)*k;
```

I Guidelines for codeing in EvtGen

This section is included for the convenience of the developers to make clear how classes, memeber functions etc. should be named. Note that a lot of the original code was written completely without any of these guidelines and hence are in complete violation of this guide. It is, however, a goal of the developers to modify the old code to be consistent with this standard.

All classes, and in general any global symbols, should start with the three letters "Evt". This is the package TLA as used in BaBar. Further class names are written using capital letters for the first letter in each word that makes up the name, e.g., EvtDecayTable.

Member functions start with a lower case letter and further words in the name are written with the first letter in capital, e.g., getDecayModel() is then the way a member function that returns the decay model should be named.

Memeber data starts with an underscore, _, and _decaymodel is then how a the member data that stores the decay models should be named.

Note in particular that it is discouraged to use an underscore in the name of either classes, member functions, or member data to separate words.

References

[1] Gangof4.

[2] D. Scora and N. Isgur, Phys. Rev. **D52**, 2783 (1995).

[3] N. Isgur, D. Scora, B. Grinstein, and M.B. Wise, Phys. Rev. **D39**, 799 (1989).

[4] The ALEPH collaboration, **CERN-PPE/97013**.

[5] S. Versillé, F. Le Diberder, CP-Fitting of $B \to 3\pi$ events (BaBar Note in preparation).

[6] 3π section of the BaBar Book.

[7] J.C. Anjos *et al.*, (E691), Phys. Rev. **D48**, 56 (1993).

[8] J.Adler *et al.*,(MARK III),Phys. Lett. **B196**, 107 (1987).

[9] J.E. Duboscq *et al.*, (CLEO) Phys. Rev. LEtt. **76**, 3898 (1996).

[10] P.Colangelo, F. De Fazio, P. Santorelli and E. Scrimieri Phys. Rev. **D53**, 3672 (1996).

[11] T.M. Aliev, M. Savci and A. Ozpineci, Phys. Rev. **D56**, 4260 (1997).

[12] I have lost this reference...

[13] I. Dunietz, H. Quinn, A. Snyder, W Toki, and H.J. Lipkin, Phys. Rev. **D43**, 2193 (1991).

[14] M. Wirbel *et al.*, Z. Phys. **C29**, 637 (1985).

[15] Reference to the PHOTOS paper.

[16] J.D. Richman, Cal. Tech. Preprint CALT-68-1148, (1984) (unpublished).

[17] M. Jacob and G. C. Wick, Ann. Phys. **7**, 404 (1959).

[18] J. D. Jackson, in *Les Houches Lectures in High Energy Physics*, 1965 C. DeWitt and M. Jacob, eds. (Goordon and Breach, New York), 1966.

Index