BAD 522 V6
EvtGen V00-11-07

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)
David Kirkby (davidk@slac.stanford.edu)
Frank Wuerthwein (fkw@fnal.gov)
Akimasa Ishikawa (akimasa@hepl.phys.nagoya-u.ac.jp)
April 11, 2006

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 B mesons, that are not implemented in the generator.

Contents

1 Introduction

There are several event generators available for simulation of particle decays in high energy physics experiments. Examples used for the simulation of B-physics include the well known packages QQ [1] and JETSET [2]. This paper describes a 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 currently collecting data, 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 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

$$B \to D^{*}\ell\nu, \qquad\qquad B \to D^{*}\ell\nu,$$
$$\quad\;\; \hookrightarrow D\pi \qquad\qquad\qquad\quad \hookrightarrow D\gamma$$

$$B \to D^{*} \quad D^{*}, \qquad\qquad B \to D^{*} \quad D^{*}.$$
$$\quad\;\; \hookrightarrow D\pi \hookrightarrow D\pi \qquad\qquad\quad \hookrightarrow D\pi \hookrightarrow D\gamma$$

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 150 classes and 25000 lines of code. There are approximately 70 models implemented that simulate a large variety of physics processes. The modularity of the code allows for easy implementation of additional models.

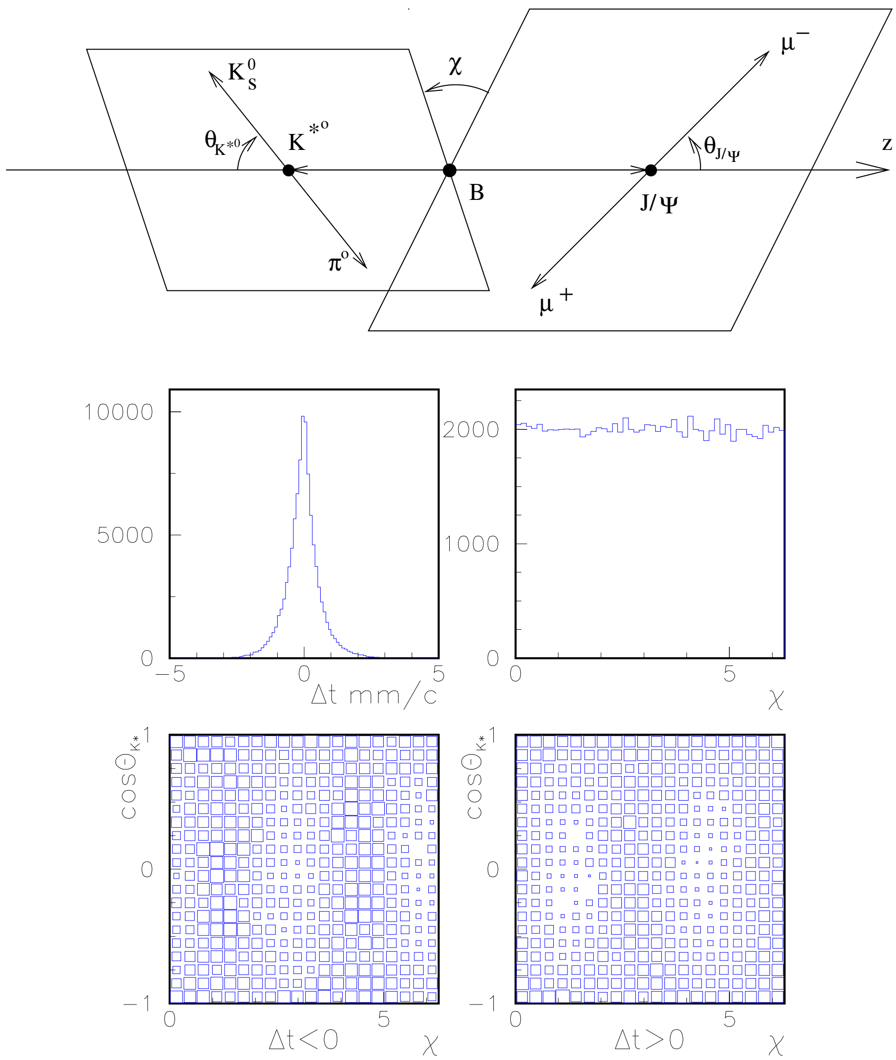

Figure 1: The top diagram defines the angles in the decay $B \to J/\psi K^*$ with $J/\psi \to \mu^+\mu^-$ and $K^{*0} \to K^0_S \pi^0$. The lower four plots show projections of the distributions from an EvtGen simulation of this decay. In this simulation the B meson was produced in an $\Upsilon(4S)$ decay and Δt is the difference in proper lifetimes of the two B mesons. The lower four plots shows (from the upper left) the Δt distribution, 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 distribution

For distribution of the EvtGen source outside of the BaBar experiment a tar file, EvtGen.tar, containing the complete source, is available. This file can be obtained from

`http://www.slac.stanford.edu/~lange/EvtGen`

Here you will find the versions that are available and short descriptions of modifications and improvements as well as known problems.

There are a handful of files other than the source files included in the `EvtGen.tar` file. The functionality of the `evt.pdl`, `DECAY.DEC`, and `Makefile` files is described below. Additionally, a `test` directory is included that provides several diagnostic tests of the EvtGen library.

`evt.pdl` is needed at runtime by EvtGen, as it contains the list of particles and particle properties to be used. This table is fully described in Section 3. Similarly, `DECAY.DEC` contains the default list of decay channels, including branching ratios, as described in Section 4.

The `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 2.1.

The directory `test` contains the files necesary to perform several tests of the EvtGen code, as detailed in Section 2.1.

2.1 Get started with EvtGen

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

First, several envirnment variables are needed to setup the external packages. The list below gives the envvar and an example setting:

1. CERN_ROOT = /cern/pro
2. ROOT_SYS = /home/lange/root
3. CLHEP_BASE_DIR = /home/lange (CLHEP in /home/lange/CLHEP and CLHEP libs in /home/lange/lib)

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:

```
lange@EvtGen>./configure
Welcome to configure for EvtGen
Deleting config.mk
This machine is running: Linux
Linux is supported.
```

```
Enter Name of c++ compiler (ie c++,cxx,CC, etc)
gcc-3.2.1
Will use gcc-3.2.1 for compilation of C++ code
Enter name of fortran comiler
f77
Will use f77 for compilation of fortran code
The EvtGen package uses photos and pythia from cernlib: Please set CERN_ROOT if not alrea
The EvtGen package uses root for histograming. Please set ROOTSYS if not already
The EvtGen package relies on CLHEP. Please set CLHEP_BASE_DIR if not already
Finding compiler includes

The gcc-3.2.1 compiler includes are
-Iignoring -Inonexistent -Idirectory -I"/usr/local/lib/gcc-lib/i686-pc-linux-gnu/3.2.1/ir
The default gcc-3.2.1 options are
-I. -DEVTSTANDALONE
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. -DEVTSTANDALONE
The default link options are
for the test program.
Please enter any others that you would like to use

Done. Type gmake lib then gmake bin
```

The `configure` script builds a file called `config.mk` that is included by the `Makefile`.

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 [2] and PHOTOS [3] packages from CERNLIB are needed, as well as the CLHEP [4] and ROOT [5] packages. A version of `lucomp.F` is included that defines additional particles used by EvtGen. The dependence on ROOT is solely for the histograming performed in the `testEvtGen` test program. If you do not need this feature it is easy to eliminate this dependence.

The most recent version has been tested using gcc-3.2.1, CERNLIB 2000, CLHEP 1.8.0.0, and ROOT 3.01.02. The only change from the default installations is that a softlink to libCLHEP.a was created to libCLHEP-g++.a. The need for this change can be eliminated by changing the `Makefile` in EvtGen.

If the library was built successfully, type `gmake bin` in order to build 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 20000
```

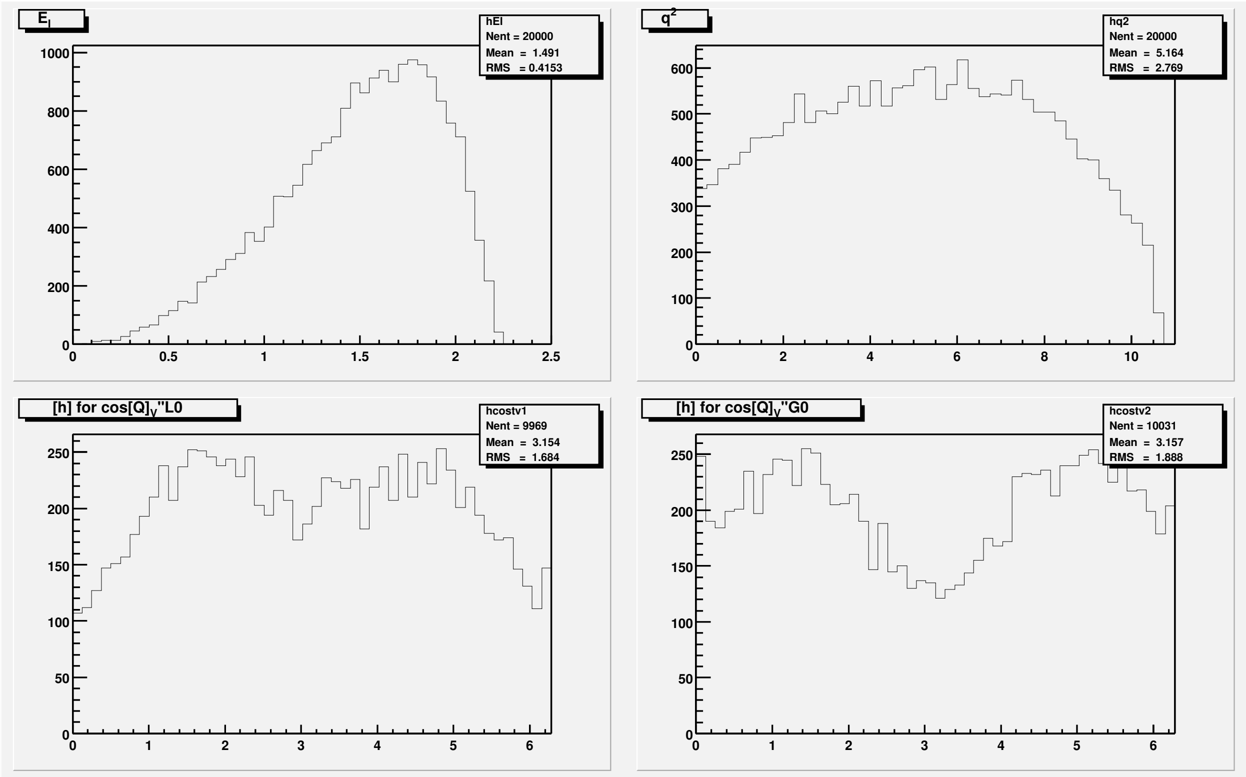

Figure 2: Histogram number 3 in `test1.root` for 1000 events shows the lepton energy in the $B \to D^* \ell \nu$ decay.

```
../testEvtGen ddalitz 20000
```

The `test` directory should now contain `test1.root` and `ddalitz.root`, each of which are ROOT files containing several histograms. Figures 2 and 3 show some of these histograms. These Figures can be reproduced using `test1-root.script` and `ddalitz-root.script`.

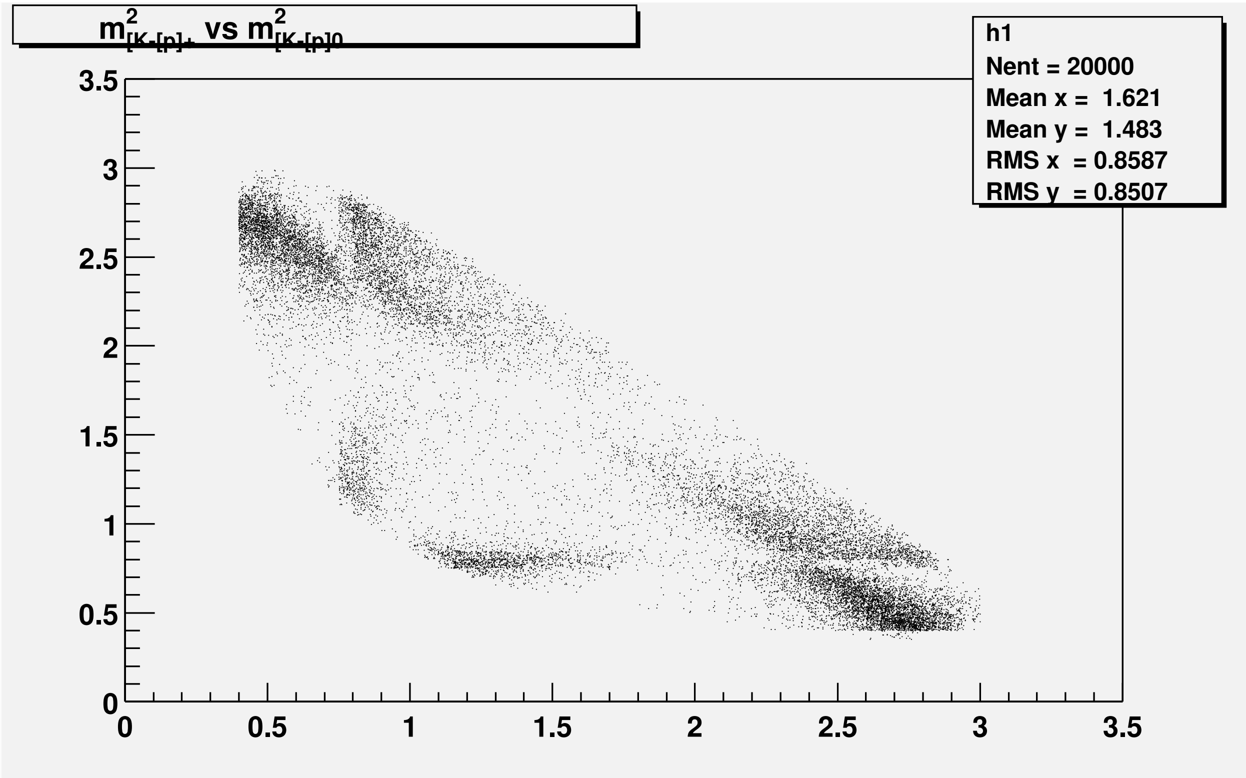

Figure 3: Histogram number 1 in `ddalitz.root` for 20000 event shows the dalitz plot for the $D \to K\pi\pi$ decay.

3 Particle properties in EvtGen

In EvtGen, all particle property information is contained within `evt.pdl` and is parsed at runtime. Each line in `evt.pdl` 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 according to the stdhep numbering scheme. The sixth through eighth columns contain the mass, width, and maximum allowed deviation from the mean mass in the downward direction, 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 mm. The twelvth column is the Lund-KC number. This is used for the interface to JetSet and has to match what is in lucomp.F.

4 Decay tables in EvtGen

The decay table included in the EvtGen package, `DECAY.DEC`, provides an extensive list of decays for resonances below the $\Upsilon(4S)$. However, the table is updated with available experimental and theoretical information on a regular basis. 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. Decay channel entries are separated by “;”. There are three main components needed to specify a decay channel. First the branching ratio is given, followed by the list of final state particles. Finally the model used to simulate the decay is specified. For many models, the order of final state particles is important. The correct order for each model can be found in the `DECAY.DEC` provided, or in Section A of this documentation. Details on available models can be found in Section A.

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 these arguments 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 and also must precede the first use of the constant that it defines. If a constant is redefined, the last value defined before the use in the decay table is used. To illustrate this consider the following example

```
Define alpha 0.9
Decay B0
1.00  pi+ pi-   SSS_CP dm alpha 1 1.0 0.0 1.0 0.0;
Enddecay
Define alpha 1.1
Decay B0B
1.00  pi+ pi-   SSS_CP dm alpha 1 1.0 0.0 1.0 0.0;
Enddecay
```

Here the decay of the B^0 will use $\alpha = 0.9$ while the $\bar{B}^0$ decay will use $\alpha = 1.1$. This means, in particular, that you can not create user decay files that change parameter values to change the default decay table settings after the default decay table has been parsed.

Once the decay channels of a particle (eg, a D^0) have been specified, the decay channels of the charge conjugate particle ($\bar{D}^0$) can be specified using the syntax

```
CDecay anti-D0
```

Another feature is that particle aliases may be defined. These aliases are useful for various reasons, including the generation of Monte Carlo for special signal channels. Consider the case of Monte Carlo for the decay $B^0 \to J/\Psi K_s^0$ with $J/\Psi \to \mu^+\mu^-$. Not all J/Ψ mesons 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

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

There is one small complication related to aliased particles involving the `CDecay` feature described above.

Sometimes it is necesary to define the charge conjugate of an alias. This is used, for example by the `CDecay` feature. To use this feature with aliases, you must specify the charge conjugate state for each alias. If you alias a particle that is its own charge conjugate, (eg, a π^0) the aliased particle will be its own charge conjugate. However, in the case of the alias `mypi+`, which represents `pi+`, a charge conjugate alais `mypi-` must be defined and you will need to tell the generator that the charge conjugate of `mypi-` is `mypi+`:

```
ChargeConj mypi+ mypi-
```

The charge conjugate of `mypi+` can not be defined to be `pi-`.

Final state radiation using the PHOTOS [3] package may be included in each decay. This option is invoked by the key word `PHOTOS`, which is placed after the list of decay daughters, but before the model name. See Appendix C for more details.

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

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

is 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. The setting of a global parameter, illustrated by `JetSetPar`, is a general feature that any decay model can implement. This is discussed further in Section 7.5.

4.1 Guidelines to write decay files

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 4.2.
- 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 `evt.pdl`.
- 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.

4.2 User decay files

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.

As the first example, consider generating $\Upsilon(4S) \to B^+B^-$, $B^+ \to D^{*0}e^+\nu_e$ 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 running EvtGen begins with parsing the full standard decay table, `DECAY.DEC`. After this, a user decay table is parsed in order to redefine the particle decays 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 B mesons. 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^+ and a B^- 100% of the time. The B^- decay is not redefined and hence it decays generically according to the entries in `DECAY.DEC`. However, the B^+ is redefined such that it is forced to decay semileptonically according to the model of ISGW2. (For more information about details about what different models do, see Appendix A.)

Another use of user decay files is to make a particle stable (if, for example, its decay is uninteresting for your purpose). To make a K_S stable do

```
Decay K0S
Enddecay
```

Anders, add b0b0bar mixing example

5 Algorithm

To illustrate how the event selection algorithm works consider the decay $B \to D^*\tau\bar{\nu}$, $D^* \to D\pi$, and $\tau \to \pi\nu$. The general case is a straight forward generalization of this example. The decay amplitude can be written as

$$A = \sum_{\lambda_{D^*}\lambda_\tau} A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau} \times A^{D^*\to D\pi}_{\lambda_{D^*}} \times A^{\tau\to\pi\nu}_{\lambda_\tau}, \tag{1}$$

where λ_{D^*} and λ_τ label the states of spin degrees of freedom of the D^* and the τ, respectively. Thus, $A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau}$ represents the decay amplitude for $B \to D^*\tau\nu$ for the six different combinations of D^* and τ states.

A possible implementation of Eq. 1 is to generate kinematics according to phase space for the entire decay chain and to calculate the probability, the amplitude squared, which is used in an accept-reject algorithm. This approach has two serious limitations. First, the maximum probability of the decay chain must be known. This is logicistally difficult given the large number of potential decay chains in B decays. Second, for long decay chains the accept-reject algorithm can be very inefficient as the entire chain must be regenerated if the event is rejected. We have implemented an algorithm that generates a decay chain as a sequence of sub-decays, thus avoiding both of these limitations.

First the decay of the B is considered. Kinematics are generated according to phase space and the probability is calculated

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

The kinematics are regenerated until the event passes an accept-reject algorithm based on P_B. After decaying the B we form the spin density matrix

$$\rho^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} = \sum_{\lambda_\tau} A^{B\to D^*\tau\nu}_{\lambda_{D^*}\lambda_\tau}[A^{B\to D^*\tau\nu}_{\lambda'_{D^*}\lambda_\tau}]^*, \tag{3}$$

which describes a D^* from the $B \to D^*\tau\nu$ decay after summing over the degrees of freedom for the τ. To generate the $D^* \to D\pi$ decay, proceed as with the B, including also ρ^{D^*}

$$P_{D^*} = \frac{1}{\mathrm{Tr}\,\rho^{D^*}} \sum_{\lambda_{D^*}\lambda'_{D^*}} \rho^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} A^{D^*\to D\pi}_{\lambda_{D^*}}[A^{D^*\to D\pi}_{\lambda'_{D^*}}]^*, \tag{4}$$

where the scale factor, $1/\mathrm{Tr}\,\rho^{D^*}$, is proportional to the decay rate, and does not affect the angular distributions. This scale factor makes the maximum decay probability of each sub-decay independent of the full decay chain.

Finally, we decay the τ. We form the density matrix

$$\tilde{\rho}^{D^*}_{\lambda_{D^*}\lambda'_{D^*}} = A^{D^*\to D\pi}_{\lambda_{D^*}}[A^{D^*\to D\pi}_{\lambda'_{D^*}}]^*, \tag{5}$$

which encapsulates the information about the D^* decay needed to properly decay the τ with the full correlations between all kinematic variables in the decay. Using the $\tilde{\rho}^{D^*}$ matrix we calculate the spin density matrix of the τ

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

As in the other decays, kinematics are generated according to phase space and the accept-reject is based on the probability calculated as in Eq. 4, replacing D^* with τ.

The algorithm was illustrated above using an example which should convey the idea. In general consider the decay

$$A \to B_1 B_2 ... B_N \tag{7}$$

where the amplitudes are denoted by

$$A^{A\to B_1 B_2 ... B_N}_{\lambda_A \lambda_{B_1} \lambda_{B_2} ... \lambda_{B_N}}. \tag{8}$$

The probability for, P_A, for this decay is given by

$$P_A = \sum_{\lambda_A \lambda'_A \lambda_{B_1} ... \lambda_{B_N}} \rho^{A}_{\lambda_A \lambda'_A} A^{A\to B_1 B_2 ... B_N}_{\lambda_A \lambda_{B_1} \lambda_{B_2} ... \lambda_{B_N}} [A^{A\to B_1 B_2 ... B_N}_{\lambda'_A \lambda_{B_1} \lambda_{B_2} ... \lambda_{B_N}}]^*. \tag{9}$$

The forward spin-density matrix ρ^{B_i}, given that B_j, $j < i$, have been decayed and have backward spin-density matrices $\hat{\rho}^{B_j}$, is given by

$$\rho^{B_i}_{\lambda_{B_i} \lambda'_{B_i}} = \sum_{\substack{\lambda_A \lambda'_A \lambda_{B_1} ... \lambda_{B_N} \\ \lambda'_{B_1} ... \lambda'_{B_{i-1}}}} \rho^{A}_{\lambda_A \lambda'_A} \hat{\rho}^{B_1}_{\lambda_{B_1} \lambda'_{B_1}} ... \hat{\rho}^{B_{i-1}}_{\lambda_{B_{i-1}} \lambda'_{B_{i-1}}} A^{A\to B_1 B_2 ... B_N}_{\lambda_A \lambda_{B_1} \lambda_{B_2} ... \lambda_{B_N}} [A^{A\to B_1 B_2 ... B_N}_{\lambda'_A \lambda'_{B_1} ... \lambda'_{B_i} \lambda_{B_{i+1}} ... \lambda_{B_N}}]^*. \tag{10}$$

After all B_i are decays the backward spin-density matrix is given by

$$\hat{\rho}^{A}_{\lambda_A \lambda'_A} = \sum_{\substack{\lambda_{B_1} ... \lambda_{B_N} \\ \lambda'_{B_1} ... \lambda'_{B_N}}} \hat{\rho}^{B_1}_{\lambda_{B_1} \lambda'_{B_1}} ... \hat{\rho}^{B_N}_{\lambda_{B_N} \lambda'_{B_N}} A^{A\to B_1 B_2 ... B_N}_{\lambda_A \lambda_{B_1} \lambda_{B_2} ... \lambda_{B_N}} [A^{A\to B_1 B_2 ... B_N}_{\lambda'_A \lambda'_{B_1} ... \lambda'_{B_N}}]^*. \tag{11}$$

6 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.[1] This section will describe how 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.

Class name	Rep.	J	States	Example
EvtScalarParticle	1	0	1	π, B^0
EvtDiracParticle	u_α	1/2	2	e, τ
EvtNeutrinoParticle	u_α	1/2	1	ν_e
EvtVectorParticle	ϵ^μ	1	3	ρ, J/Ψ
EvtPhotonParticle	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 Pauli-Dirac convention for the gamma matrices, as discussed in Section 8.4. 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 handed neutrinos and right handed anti-neutrinos are considered.

The complex 4-vector ϵ^μ is 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 and are implemented in the `EvtTensorParticle` class.

For each particle initialized in EvtGen, a set of basis states is created, where the number of basis states is the same as the number of spin degrees of freedom. 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 states in the parent's rest frame are available. As an example, consider the basis for a massive spin 1 particle in it's own rest frame

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

[1]Other particles will be added as need arises.

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

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

where $g^{\mu\nu} - p^{\mu}p^{\nu}/m^2$ is the propagator for an on-shell spin 1 particle.

In order to write a new decay model, there is no reason to need to know the exact choice of these basis states. The code needed to describe the amplitude for a decay process is independent of these states, as long as they are complete, orthogonal and normalized.

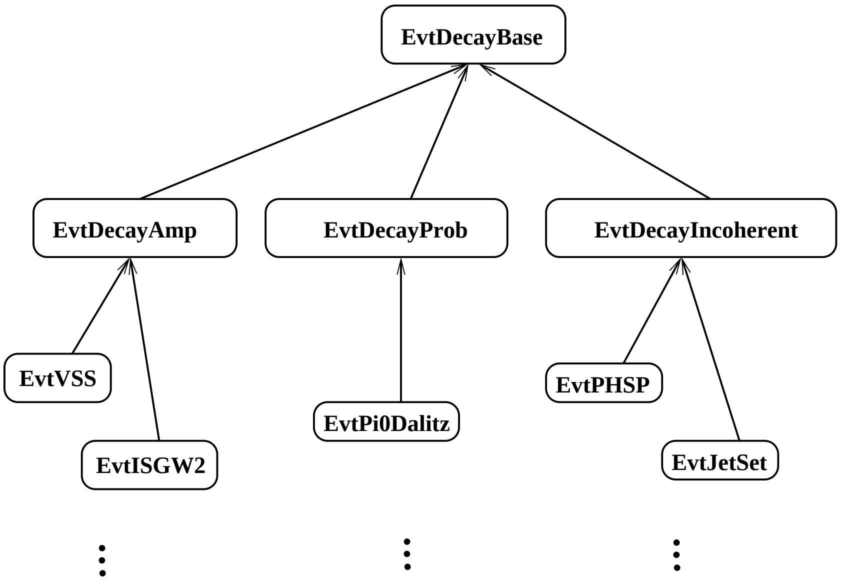

Figure 4: Diagram of the `EvtDecayBase` class and classes derived from it. `EvtDecayAmp`, `EvtDecayTBaseProb`, and `EvtDecayIncoherent` are templated classes that are used by modules that compute the full decay amplitude, that compute the decay probability, and those that return unpolarized daughters,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

7 Decay models

7.1 Introduction to decay models

Anders put text here...

Each decay model is a class that 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. An example of 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 $B \to \pi^+\pi^-\pi^+\pi^-$.

7.2 Creating new decay models

To simplify the writing of decay models there are three different classes that are derived from `EvtDecayBase`. These are `EvtDecayAmp`, `EvtDecayProb`, and `EvtDecayIncoherent`. When writing decay models it is most convenient to derive from one of these classes. These classes have slightly different interfaces depending on what information the decay model provides. Some models will provide the complete amplitude information where as other models might provide just a probability or a set of four-vectors. Models that don't provide the full set of amplitudes will of course not be able to simulate the complete angular distributions. Below, a short description is given to explain when you want to use each of these classes as the base class for your decay model.

- `EvtDecayAmp` allows you to specify the complete amplitudes and hence do a complete simulation of the angular distributions.
- `EvtDecayProb` allows you to calculate a probability for the decay. This probability is then used in the accept-reject method. Any spin information is lost, all produced particles are unpolarized and uncorrelated.
- `EvtDecayIncoherent` just accepts the four vectors as generated. This is most useful when interfacing to another generator, e.g. JetSet. Any spin information is lost, all produced particles are unpolarized and uncorrelated.

7.3 Example: EvtVSS model

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

```
//--------------------------------------------------------------------------
//
// 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/EvtDecayAmp.hh"
#include "EvtGen/EvtParticle.hh"

class EvtVSS:public  EvtDecayAmp  {

public:
```

```
  EvtVSS() {}
  virtual ~EvtVSS();

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

  void init();
  void initProbMax();
  void decay(EvtParticle *p);
};

#endif
```

The constructor and destructor in this example are emtpy. The virtual destructor must be implemented in the .cc file, and is not shown.

The simplest method to implement is the `clone()` method, which is used to create an instance of a model each time the it is used in the decay table. This is done using the prototype pattern [6]. The implementation of the `clone()` method in the `EvtVSS` module is

```
EvtDecayBase* EvtVSS::clone(){
  return new EvtVSS;
}
```

The `getName` method specifies the name of the module, and 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 in any decay. 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, a warning is printed.

When particles of finite width are present in the decay tree, it may be difficult to efficiently set the maximum probability, as the true maximum may depend on the mass of the wide particle. When this is the case, the model will still give correct output. However it will be very inefficient and may get into semi-infinite loops for extreme mass configurations. Care should be taken in the implementation of decay amplitudes so that this is not the case.

The maximum probability is set by calling `setProbMax(prbmax)` method, which is a member function of `EvtDecayBase`.

```
void EvtVSS::initProbMax() {
   setProbMax(1.0);
}
```

Models that can handle many different decays may have different maximal probabilities depending on what initial and final state is being generated. To accomadate such models, the parent and daughter information (`ndaug, parent, narg, daug, args` from the `EvtDecayBase` base class) can be accessed to calculate the maximum probabilites.

During development of a model it is often convenient not 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 when performing 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 EvtGen that given the random number seed at the start of the event the same decay must be obtained independently of previous events.

The `init` method is a generic initialization function that will be called once for each occurance of the decay model in the decay file. The `init` function allows modules to precompute quantities, for example 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 consistency between the number of arguments in the decay table and that expected by the model. The `init` method for the `EvtVSS` class is shown below.

```
void EvtVSS::Init(){

  // check that there are 0 arguments

  if (getNArg()!=0) {

    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " 0 arguments but found:"<<getNArg()<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    ::abort();

  }
  if ( getNDaug()!=2) {
    report(INFO,"EvtGen") << getNDaug() <<" "<<EvtPDL::name(getDaug(0))<<endl;
    report(ERROR,"EvtGen") << "EvtVSS generator expected "
                           << " a 2 daughters, found:"<<
                           getNDaug()<<endl;
    report(ERROR,"EvtGen") << "Will terminate execution!"<<endl;
    ::abort();
  }

  EvtSpinType::spintype parenttype = EvtPDL::getSpinType(getParentId());
  EvtSpinType::spintype d1type=EvtPDL::getSpinType(getDaug(0));
  EvtSpinType::spintype d2type=EvtPDL::getSpinType(getDaug(1));
```

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


}
```

Finally, we describe the `decay` member function. This method does the actual work of decaying the particle, including the generation of kinematics as well as the amplitude or probability calculation. 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->initializePhaseSpace(getNDaug(),getDaugs());

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

  double norm=1.0/pdaug.d3mag();

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

  return;
}
```

The argument of 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 daughter particles and to link these particles onto the decay tree. This is done by the `EvtParticle::initializePhaseSpace` member function. The arguments to this method are the number and list of daughters, as specified in the decay table. The `initializePhaseSpace` function generates kinematics according to phase space.

For models that use the class `EvtDecayAmp`, 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 using the `vertex` member function. The first argument is the index of the basis vector used to compute the amplitude. The `EvtDecayAmp` header file 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)
```

After computing the amplitude, the `decay` function returns and the amplitudes are used in an accept-reject test. If failed, the `decay` method is called once more and a new kinematic configuration is generated and new amplitudes are calculated.

7.4 Example: EvtPi0Dalitz

For models which derive from `EvtDecayProb` or `EvtDecayIncoherent` classes, the `decay` function is slightly different. As an example of a model derived from the `EvtDecayProb` 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){

  EvtParticle *ep, *em, *gamma;

  p->makeDaughters(getNDaug(),getDaugs());
  ep=p->getDaug(0);
  em=p->getDaug(1);
  gamma=p->getDaug(2);



  double mass[3];

  double m = p->mass();

  findMasses( p, getNDaug(), getDaugs(), mass );

  EvtVector4R p4[3];

  setWeight(EvtGenKine::PhaseSpacePole
           (m,mass[0],mass[1],mass[2],0.00000002,p4));

  ep->init( getDaug(0), p4[0] );
  em->init( getDaug(1), p4[1]);
  gamma->init( getDaug(2), p4[2]);

  //ep em invariant mass^2
  double m2=(p4[0]+p4[1]).mass2();
  EvtVector4R q=p4[0]+p4[1];
  //Just use the prob summed over spins...

  EvtTensor4C w,v;

  v=2.0*(p4[2]*q)*directProd(q,p4[2])
    - (p4[2]*q)*(p4[2]*q)*EvtTensor4C::g()
    -m2*directProd(p4[2],p4[2]);

  w=4.0*( directProd(p4[0],p4[1]) + directProd(p4[1],p4[0])
   -EvtTensor4C::g()*(p4[0]*p4[1]-p4[0].mass2()));
```

```
  double prob=(real(cont(v,w)))/(m2*m2);
  prob *=(1.0/( (0.768*0.768-m2)*(0.768*0.768-m2)
           +0.768*0.768*0.151*0.151));

  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 EvtDecayIncoherent class.

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

  p->initializePhaseSpace(ndaug,daug);
  return ;
}
```

In this case, the decay function must only 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 momenta and spin basis vector information for a given particle. This is discussed in Section 9.

7.5 Model parameters

In Section 4 the feature of parameters for a model was introduced. In particular, JetSetPar is an example of the mechanism that allows any model to set parameters. To make use of this, the methods

```
EvtString commandName();
void command(EvtString cmd);
```

must be implemented within the model. The commandName method returns a string that contains the identifier to be searched for in the decay table. In the case of the JetSet model, this was JetSetPar. It is encouraged that this string start with the model name, such that it is obvious to which model the parameter belongs. When the decay table is parsed, the method command is called with the string found in the decay table as the argument. This function is invoked on the prototype object. Thus, any data that is obtained through this mechanism should be stored in a static variable such that it is common to all instances of the model.

8 Conventions

This section discusses the conventions we use for various physics quantities in the code.

8.1 Units

In EvtGen, $c = 1$, such that mass, energy and momentum are all measured in units of GeV. Similarly, time and space have units of mm.

8.2 Four-vectors

There are two types of four-vectors used in EvtGen, `EvtVector4R` and `EvtVector4C`, which are real and complex respectively.

A four-vector is represented by $p^\mu = (E, \vec{p})$. When a four-vector is used its components are always corresponding to raised indices. A contraction 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 a particle with four-momentum p.

8.3 Tensors

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

8.4 Dirac spinors

Dirac spinors are represented as a 4 component spinor in the Dirac-Pauli representation, with initial state fermions or final state anti-fermions.

8.5 Gamma matrices

Dirac gamma matrices are also 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

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

9 Classes

This section describes the classes in the EvtGen package, except for the classes are models; these are described separately in Appendix A.

9.1 EvtAmp

This class keeps track of the amplitudes computed by the decay models. It provides member functions to calculate spin-density matrices from the amplitudes, as described in section 5. The only member functions that a user writing decay models should need to use are the `vertex` functions that allow you to set the amplitudes that are calculated in a decay model

- `void vertex(const EvtComplex& amp)`
  Sets the amplitude in a decay where all particles, both in the initial and final state, have only one spin degree of freedom.

- `void vertex(int i1, const EvtComplex& amp)`
  Sets the amplitude when one particle has more than one spin degree of freedom.

- `void vertex(int i1, int i2, const EvtComplex& amp)`
  Sets the amplitudes in a decay with two particles having non-trivial spin.

- `void vertex(int i1, int i2, int i3, const EvtComplex& amp)`
  Sets the amplitudes in a decay with three particles having non-trivial spin.

- *there is also a general case – Need to fix and document*

The other functions are for internal use only. Some of the core functionality of EvtGen is implemented in these functions that perform the low level manipulation of the amplitudes and spin-density matrices.

- `EvtAmp()`
  Default constructor.

- `void init(EvtId p,int ndaug,EvtId *daug)`
  Initializes amplitude for the particles specified.

- `void setNDaug(int n)`
  Sets the number of daughters.

- `void setNState(int parent_states,int *daug_states)`
  Sets the number of spin states each particle has.

- `void setAmp(int *ind,const EvtComplex& amp)`
  Sets an amplitude.

- `const EvtComplex& getAmp(int *ind)const`
  Returns an amplitude.

- `EvtSpinDensity getSpinDensity()`
  Returns spin density matrix for the parent.

- `EvtSpinDensity contract(int i,const EvtAmp& a)`
  Contracts amplitude to calculate spin-density matrix.

- `EvtAmp contract(int i, const EvtSpinDensity& rho)`
  Contracts amplitude with spin-density matrix to calculate new amplitude.

- `EvtSpinDensity getForwardSpinDensity(EvtSpinDensity *rho_list,int k)`
  Calculates forward spin-density matrix for one of the daughters.

- `EvtSpinDensity getBackwardSpinDensity(EvtSpinDensity *rho_list)`
  Contracts amplitudes to calculate the backward spin-density matrix for the parent.

9.2 EvtCPUtil

Update with Anders fixes the code to be more general

This class contains some utilities that are useful for generating CP-violating decays from the $\Upsilon(4S)$ system. In particular, it contains the two methods

- `static void OtherB(EvtParticle *p, double &t, EvtId &otherb)`
  Particle `p` is a B meson from an $\Upsilon(4S)$ decay. `otherb` is the flavor that you want the other B to have. When this function is invoked, flavor of the other B is changed, such that it becomes of type `otherb`. The lifetime of the other B meson is returned in `t`.

- `static void OtherB(EvtParticle *p, double &t, EvtId &otherb,`
  `double probB0)`
  Similar to the function above, except that a probability is supplied for how ofthe th other B should be a B^0.

9.3 EvtComplex

Using the implementation of complex numbers provided by the compiler has caused constant problems with porting EvtGen to different platforms, as these implementations do not generally conform to a uniform standard. Therefore, we have implemented the `EvtComplex` class. This implemention is not complete.

David should add the currently implemented function.

9.4 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            =  2.99792458E11; // mm/sec
```

9.5 EvtDecayBase

This class is the base class for decay models and contains the interface for the decay models to the framework. The most important member function is `decay(EvtParticle *)`, which performs the actual decay of the particle. Before explaining the purpose of other member function, recall that for each entry in the decay table an instance of the decay model class is created.

There are three classes that derive from `EvtDecayBase`; `EvtDecayAmp`, `EvtDecayProb`, and `EvtDecayIncoherent`. These classes proved slightly different interfaces for writing decay models, as discussed in detail in Section 7

The `EvtDecayBase` class provides the following member fuctions:

- `virtual void getName(EvtString& name)=0`
  This function must be implemented each decay model. `getName` returns the name of the model, to be used in the decay table.
- `virtual EvtDecayBase* clone()=0`
  This is a pure virtual function, has to be implemented in the decay model, it returns a new instance of the derived class.
- `virtual void Decay(EvtParticle *p)=0`
  This method implements the actual decay. Each decay model should implement this function.
- `virtual void makeDecay(EvtParticle *p)=0`
  Implemented in the `EvtDecayAmp`, `EvtDecayProb`, and `EvtDecayIncoherent` classes. Implements the accept reject as appropriate for the type of decay model. In the `EvtDecayAmp` implementation, the decay probability is computed from amplitudes, which are set by the decay models.
- `void disableCheckQ()`
  By default, EvtGen checks each decay for charge conservation. Sometimes this is not appropriate, for example with the `SINGLE` model, which generates single particles. If `disableCheckQ()` is called in the `init()` member function, this check will not be performed.

- `void init()`
  If the model to be implemented has any data members that must be initialized this should be done in the call to the `init` member function. An example of such initialization might be to process the arguments into some more convienient form.

- `void initProbMax()`
  `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()`. The maximum probability is set via the `setProbMax(double )` member function.

9.6 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 using the standard Pauli-Dirac convention for the γ-matrices. 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 some common matrix elements as summarized in Table 2.

Matrix elemet	Function	Return type
$\langle\bar{u}\|\gamma^\mu(1-\gamma^5)\|v\rangle$	`EvtLeptonVACurrent(u,v)`	`EvtVector4C`
$\langle\bar{u}\|\gamma^\mu\|v\rangle$	`EvtLeptonVCurrent(u,v)`	`EvtVector4C`
$\langle\bar{u}\|\gamma^\mu\gamma^5\|v\rangle$	`EvtLeptonACurrent(u,v)`	`EvtVector4C`
$\langle\bar{u}\|v\rangle$	`EvtLeptonSCurrent(u,v)`	`EvtComplex`
$\langle\bar{u}\|\gamma^5\|v\rangle$	`EvtLeptonPCurrent(u,v)`	`EvtComplex`
$\langle\bar{u}\|\sigma^{\mu\nu}\|v\rangle$	`EvtLeptonTCurrent(u,v)`	`EvtTensor4C`

Table 2: This table sumarizes the methods in `EvtDiracSpinor` that evaluates special matrix elements.

Also see Section 9.7 for operations between Dirac spinors and γ-matrices.

9.7 EvtGammaMatrix

EvtGammaMatrix is a class for handling complex 4×4 matrices. The generic operations that are available are addition (+), subtraction (-), and multiplication (*). An EvtGammaMatrix can also be multiplied by a complex scalar. Besides these generic operations on matrices, there are special operations related to gamma matrices. There are static member functions that returns the gamma matrices according to the standard Pauli-Dirac representaion. Thr g0(), g1(), g2(), g3(), g5(), and id() functions return $\gamma^0, \gamma^1, \gamma^2, \gamma^3, \gamma^5$ and the identity matrix, respectively. Besides these basic functions there are four more specialized functions, va0(), va1(), va2(), and va3() that are used in the evaluation of the weak lepton current. These matrices are given by $\gamma^0\gamma^\mu(1-\gamma^5)$, where $\mu = (0,1,2,3)$. Similarly, the functions v0(), v1(), v2(), and v3() return $\gamma^0\gamma^\mu$, where $\mu = (0,1,2,3)$.

There is also an operation (*) defined, which multiplies an EvtGammaMatrix with an EvtDiracSpinor, and returns an EvtDiracSpinor.

9.8 EvtGen

This class provides the interface to EvtGen for an external user, see Appendix B.

9.9 EvtGenKine

EvtGenKine contains tools for generating kinematics such as phase space distributions. Currently only two functions are implemented

- static double PhaseSpace(int ndaug, double mass[10], EvtVector4R p4[10], double mp)
  Generates phase space for a decay with ndaug daughters with masses given by mass for a parent with mass mp. The generated four-vectors are stored in p4.

- static double PhaseSpacePole(double M, double m1, double m2, double m3, double a,EvtVector4R p4[10])
  Similar to the PhaseSpace member function, expect that the kinematics are generated with a $1/m$ pole in the invariant mass of particle 1 and 2 in a three-body decay, (for example, $\pi^0 \to e^+e^-\gamma$). a is the strength of the pole. This method is designed to speed up the generation of decays with a pole in the decay distribution. The return value of this function is needed to correctly handle the accept-reject algorithm, and must be passed to the EvtDecayBase framework via the void setWeight(double) member function. For example:

  ```
  setWeight(EvtGenKine::PhaseSpacePole
     (m,mass[0],mass[1],mass[2],0.00000002,p4));
  ```

  is used in the EvtPi0Dalitz model.

9.10 EvtId

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

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.

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 defined 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 `!=` compare the `_id` when determining if two particles are the same. Therefore `mypi0` and `pi0` are considered the same particles, when compared using the `==` operator.

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, so that so that we allow different decays for an aliased particle and its alias. 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. However, for aliased particles, the charge conjugate must be specified by hand. See Section 9.14 and 4.

9.11 EvtKine

This class provides some utility functions for calculating kinematic quantities such as decay angles. See Appendix G.

9.12 EvtLineShape

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

- `static double BreitWigner( double mass,`
  `double width, double min, double max)`

9.13 EvtModel

This class handles the registration of decay models.

9.14 EvtPDL

The particle information read from the `evt.pdl` 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 particles in the list, not including aliases.

Except for the aliases the `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.

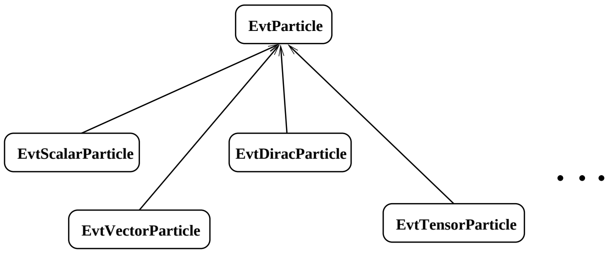

Figure 5: `EvtParticle` forms the base class for the concrete implementations of the different particle types in EvtGen.

9.15 EvtPHOTOS

Provides an interface to the PHOTOS package for generation of final state radiation.

- `static void PHOTOS(EvtParticle *p)`

If PHOTOS generates additional photons they are added at the end of the list of particles produced.

9.16 EvtPartProp

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

9.17 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 3. 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.

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

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 3: This table shows the accessor functions for the basis vectors.

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 in `EvtParticle`:

- `EvtParticle* getDaug(int i)`
  Returns a pointer to the i: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.

9.18 EvtParticleDecay

Stores information for one particle decay. This class is not used yet.

9.19 EvtParticleDecayList

Stores the list of decays of one particle. This class is not used yet

9.20 EvtParticleNum

Defines `EvtID` for all particles.

9.21 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.

9.22 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. The interface is some what clumsy.

- `static double Flat( double min, double max )`
  Generate pseudo random nmber between `min` and `max`.

- `static double Flat( double max )`
  Generate pseudo random nmber between 0 and `max`.

- `static double Flat( )`
  Generate pseudo random nmber between 0 and 1.

- `static double random()`
  Generate pseudo random nmber between 0 and 1.

- `static RandFcnD& ranFcnD()`
  Return reference to random function.

- `static RandFcnF& ranFcnF()`
  Return reference to random function.

- `static HepRandomEngine** ranEngine()` Return the HepRandomEngine.

- `static void setRandFcnD(RandFcnD& arandfcnd)`
  Use a random function that is a static function returning a double precision number.

- `static void setRandFcnF(RandFcnF& arandfcnf)`
  Use a random function that is a static function returning a double single number.

- `static void setRandEngine(HepRandomEngine& aHepRandomEngine)`
  Use a CLHEP random engine.

9.23 EvtReadDecay

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

9.24 EvtReport

Utility to print out mesages from EvtGen.

9.25 EvtResonance

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:

$$c_k e^{i\theta_k} F_k^{BW}(x,y) D_k^{ang}(x,y) \tag{21}$$

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:

$$c\, e^{i\theta} \sqrt{\frac{\Gamma}{2\pi}} \left(\frac{1}{(m_{12} - m_{BW}) - \frac{i\Gamma}{2}} \right) \tag{22}$$

For spin 1 the amplitude is:

$$c\,e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{cos\phi_3}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{23}$$

For spin 2:

$$c\,e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{\frac{3}{2}cos^2\phi_3-\frac{1}{2}}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{24}$$

For spin 3:

$$c\,e^{i\theta}\sqrt{\frac{\Gamma}{2\pi}}\left(\frac{\frac{5}{2}cos^3\phi_3-\frac{3}{2}cos\phi_3}{(m_{12}-m_{BW})-\frac{i\Gamma}{2}}\right) \tag{25}$$

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:

$$BW(m^2_{ij}) = \frac{\sqrt{\Gamma_0}\,M}{(m^2_R - m^2_{ij}) - i\Gamma m_R} \tag{26}$$

where the matrix element M is:

$$M = (P_S - P_k)^\mu (g^{\mu\nu} - P^\mu_{ij}P^\nu_{ij}/m^2_R)(P^2_i - P^2_j) \tag{27}$$

$$|M|^2 = (m^2_{ik} - m^2_{jk} - (m^2_S - m^2_k)(m^2_i - m^2_j)/m^2_R)\frac{1+(Rp_{jR})^2}{1+(Rp_j)^2} \tag{28}$$

$$\Gamma = \Gamma_0 \frac{m_R}{m_{ij}}\left(\frac{p_j}{p_{jR}}\right)^3 \frac{1+(Rp_{jR})^2}{1+(Rp_j)^2} \tag{29}$$

Here, m_R and Γ_0 are the mass and width of the m^2_{ij} 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 $m^2_{ij} = m^2_R$. 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.

9.26 EvtSecondary

Allows EvtGen not to write secondary particles to StdHep. This class will most likely be removed.

9.27 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 freedom of a spin 2 particle.) Functions are provided to manipulate the components of the spin density matrix as well as to calculate probabilites.

9.28 EvtSpinType

Defines the folowing enum for the different particle types that EvtGen handles.

```
enum spintype { SCALAR,VECTOR,TENSOR,DIRAC,PHOTON,NEUTRINO,STRING };
```

9.29 EvtStdHep

This class flattens out the EvtGen decay tree that is used internaly to represent the particles and stores the particles in a structure that is parallel to StdHep.

9.30 EvtString

This class is used by EvtGen to represent character strings. It does not provide a full interface for what you might need to do with strings. But the following is available, other things can be added if need arise.

- `EvtString()`
  Default constructor.

- `EvtString(const char* cptr)`
  Constructor from char*.

- `EvtString(const EvtString& string)`
  Copy constructor.

- `friend ostream& operator<<(ostream& s, const EvtString& str)`
  Print out string.

- `friend istream& operator>>(istream& s, EvtString& str)`
  Read in string.

- `virtual  EvtString()`
  Virtual destructor.

- `EvtString operator=(const EvtString& string)`
  Assignment operator.

- `int operator==(const EvtString& string) const`
  Equal operator between two strings.

- `int operator!=(const EvtString& string) const`
  Not equal operator between two strings.

- `int operator==(const char* string)`
  Equal operator between EvtString and char *.

- `int operator!=(const char* string)`
  Not equal operator between EvtString and char *.

- `char* value() const`
  Return pointer to a char*.

9.31 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.

9.32 EvtTemplateDummy

This class was introduced just such that the EvtGen package made use of templates, this should be removed.

9.33 EvtTensor3C

Complex rank 2 tensors in 3 dimensions.

9.34 EvtTensor4C

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 `EvtTensor4C` 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=directProd(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 `EvtTensor4C::g()`.

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

9.35 EvtVector3C

Complex three-vectors.

9.36 EvtVector3R

Real three-vectors.

9.37 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.

9.38 EvtVector4R

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

The operators =, +, -, +=, -= are supported with the obvious meaning, also multiplication and divison by real numbers are available.

The quantity $p^\mu p_\mu$ can be evaluated using the `mass2()` member function and $\sqrt{p^\mu p_\mu}$, the mass, is given by `mass()`. The member function `d3mag()` evaluates the magnitude of the spatial components, i.e., the three momentum.

Components of a four-vector can be manipulated with the `set(int i, double d)` member function which sets component `i` of the vector to `d`. The components are labeled from 0 to 3. O is the time component and 1 through 3 are the space components. To set all components at once the `set(double t, double x, double y, double z)` memeber function can be used. `get(int i)` accesses the `i`:th component of the vector.

The `EvtVector4R boost_to_v4(const EvtVector4R& v2,const EvtVector4R& vto)` function returns the four vector `v2` boosted to the frame in which a particle with momentum `vto` is at rest.

10 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.

10.1 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:

$$|B^0_{\rm phys}(t)\rangle = e^{-\Gamma t/2} e^{i\overline{m}t}\left[|B^0\rangle\cos\left(\frac{\Delta m t}{2}\right) + i\frac{q}{p}|\bar{B}^0\rangle\sin\left(\frac{\Delta m t}{2}\right)\right], \tag{30}$$

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:

$$P(B^0_{\rm phys} \to B^0) = \frac{1}{2}e^{-\Gamma t}(1+\cos(\Delta m t)) \tag{31}$$
$$P(B^0_{\rm phys} \to \bar{B}^0) = \frac{1}{2}e^{-\Gamma t}(1-\cos(\Delta m t)) \tag{32}$$
$$\tag{33}$$

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

$$P(B^0_{\rm phys} \to B^0) = \frac{x_d^2}{2\,(1+x_d^2)} \tag{34}$$

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

$$\tag{36}$$

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:

$$\frac{N(\Upsilon(4S) \to B^0 B^0) + N(\Upsilon(4S) \to \bar{B}^0 \bar{B}^0)}{N(\Upsilon(4S) \to B^0 \bar{B}^0)} = \frac{x_d^2}{2 +\, x_d^2}. \tag{37}$$

Currently, mixing is implemented in two decay models: `VSS_MIX` and `VSS_BMIX`. These models both take the mass difference to be a parameter and use the width assigned in the particle properties table, i.e., from the evt.pdl file in the BaBar environment. These models differ in the number of parameters that the user must supply. The `VSS_MIX` model requires that the branching fractions into mixed and unmixed final states be set by hand, and does not require that they be consistent with the mass difference and lifetime. The $\texttt{VSS}_\texttt{B}\texttt{MIX}$ model calculates the branching fractions automatically and is therefore safer and easier to use. One can use this model as follows:

```
Decay Upsilon(4S)
0.5  B+       B-       VSS;
0.5  B0       anti-B0  VSS_BMIX;
Enddecay
```

Both models give identical results when the `VSS_MIX` branching fractions are set correctly.

10.2 Mixing and Non-zero Lifetime Differences.

This section should probably be merged with the previous but is currently written separately, mostly due to historical reasons, as this is only needed for B_s mixing, which was added later.

In this section B^0 and $\bar{B}^0$ will generically denote any neutral meson flavor eigenstate. These states are not eigenstates of the full hamiltonian and hence do not have a definite mass or lifetime. The states that are eigenstates of the hamiltonian are

$$|B_L\rangle = p|B^0\rangle + q|\bar{B}^0\rangle, \tag{38}$$
$$|B_H\rangle = p|B^0\rangle - q|\bar{B}^0\rangle. \tag{39}$$

The states B_L and B_H have masses, m_L and m_H, and widths, Γ_L and Γ_H, respectively such that

$$|B_{L,H}(t)\rangle = e^{-(\Gamma_{L,T}/2 + i m_{L,T})t}|B_{L,H}\rangle \tag{40}$$

where $B_{L,H}(t)$ is the state after evolving the state $B_{L,H}$ for a time t.

This allows us to evaluate several matrix elements that are useful for calculating the mixing rates

$$\begin{aligned}
\langle B^0|B^0(t)\rangle &= \frac{1}{2p}(\langle B^0|B_L(t)\rangle + \langle B^0|B_H(t)\rangle) \\
&= \frac{1}{2}(e^{-(\Gamma_L/2+im_L)t} + e^{-(\Gamma_H/2+im_H)t}), \qquad (41) \\
\langle \bar{B}^0|B^0(t)\rangle &= \frac{1}{2p}(\langle \bar{B}^0|B_L(t)\rangle + \langle \bar{B}^0|B_H(t)\rangle) \\
&= \frac{q}{2p}(e^{-(\Gamma_L/2+im_L)t} - e^{-(\Gamma_H/2+im_H)t}), \qquad (42) \\
\langle B^0|\bar{B}^0(t)\rangle &= \frac{1}{2q}(\langle B^0|B_L(t)\rangle - \langle B^0|B_H(t)\rangle) \\
&= \frac{p}{2q}(e^{-(\Gamma_L/2+im_L)t} - e^{-(\Gamma_H/2+im_H)t}), \qquad (43) \\
\langle \bar{B}^0|\bar{B}^0(t)\rangle &= \frac{1}{2q}(\langle \bar{B}^0|B_L(t)\rangle + \langle \bar{B}^0|B_H(t)\rangle) \\
&= \frac{1}{2}(e^{-(\Gamma_L/2+im_L)t} + e^{-(\Gamma_H/2+im_H)t}). \qquad (44)
\end{aligned}$$

Now it is straight forward to calculate the mixing rate. Let χ define the probability that a meson produced, at $t=0$, as a B^0 will decay as a $\bar{B}^0$. It is now easy to see that

$$\begin{aligned}
\chi &= \frac{\int_0^\infty |\langle \bar{B}^0|B^0(t)\rangle|^2 dt}{\int_0^\infty |\langle \bar{B}^0|B^0(t)\rangle|^2 dt + \int_0^\infty |\langle B^0|B^0(t)\rangle|^2 dt} \\
&= \frac{x^2+y^2}{x^2+y^2+|\frac{p}{q}|^2(2+x^2-y^2)} \qquad (45)
\end{aligned}$$

where

$$x \equiv \frac{\Delta m}{\Gamma}, \qquad y \equiv \frac{\Delta\Gamma}{\Gamma} \qquad (46)$$

and

$$\Gamma \equiv \frac{\Gamma_L+\Gamma_H}{2}, \qquad \Delta\Gamma \equiv \Gamma_H - \Gamma_L, \qquad \Delta m \equiv m_H - m_L. \qquad (47)$$

Similarly the probability that that a meson produced, at $t=0$ as a $\bar{B}^0$ will decay as a B^0 is given by

$$\begin{aligned}
\bar{\chi} &= \frac{\int_0^\infty |\langle B^0|\bar{B}^0(t)\rangle|^2 dt}{\int_0^\infty |\langle B^0|\bar{B}^0(t)\rangle|^2 dt + \int_0^\infty |\langle \bar{B}^0|\bar{B}^0(t)\rangle|^2 dt} \\
&= \frac{x^2+y^2}{x^2+y^2+|\frac{q}{p}|^2(2+x^2-y^2)}. \qquad (48)
\end{aligned}$$

In the limit of $|p/q|=1$ and $y=0$ we have

$$P_\pm(t) = \chi = \bar{\chi} = \frac{x^2}{2(1+x^2)} \qquad (49)$$

as expected. The time distribution, arbitrary normalization, is given by

$$e^{-\Gamma_L t}(1 + e^{\Delta\Gamma t} \pm 2e^{\Delta\Gamma t/2}\cos\Delta m t) \tag{50}$$

The minus sign corresponds to the case of mixing, and the plus sign to the case of no mixing.

10.3 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 alpha dm CP |A| arg(A) |Abar| arg(Abar);
Enddecay
End
```

The model used in this example is SSS_CP. Its first argument is alpha, is the relevant CKM angle (in this case it is α); the second argument dm is the mass difference of the two B mass eigenstates; 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:

$$A_f(t) = \frac{\Gamma(B^0(t) \to f) - \Gamma(\overline{B}^0(t) \to f)}{\Gamma(B^0(t) \to f) + \Gamma(\overline{B}^0(t) \to f)} \tag{51}$$

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:

$$fr \equiv \frac{\int_{-\infty}^{\infty} \Gamma(\overline{B}(t)^0 \to f)dt}{\int_{-\infty}^{\infty} \left[\Gamma(B^0(t) \to f) + \Gamma(\overline{B}^0(t) \to f)\right] dt} \tag{52}$$

so that

$$fr = \frac{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right)}{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right) + |A_f|^2 \left(1 + |r_f|^2 + \frac{(1-|r_f|^2)}{1+x_d^2}\right)}, \tag{53}$$

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 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 "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.

10.4 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.

10.5 Unified implementation of CP-violating and mixing

This section describes a *proposal* for how to unify and simplify the simulation of CP-violating decays and mixing in EvtGen, this should allow writing models that works independently of how the neutral B meson is produced, i.e. in a coherent pair in an $\Upsilon(4S)$ decay or incoherently e.g. in a $p\bar{p}$ collision.

10.5.1 CP violation

The decays that we are considering here are common final states to a neutral B meson and its antiparticle, e.g. $B \to J/\psi K_S$ or $B \to J/\psi K^*$. At the $\Upsilon(4S)$ in the absence of direct CP violation these decays can be simplified as we can make the assumption that you have an equal number of B^0 and $\bar{B}^0$ tags, or equivalent that there is no time integrated asymmetry. However, we want to write the code such that no assumptions are made about the numbers of B^0 and $\bar{B}^0$ tags. For B mesons that are produced incoherently we always have to consider the asymmetries in the rates for an initial B^0 or $\bar{B}^0$.

To allow generating these asymmetries correctly these models will be allowed to modify the flavor of the B meson that is decayed. For the $\Upsilon(4S)$ this means changing the the flavor of 'the other' B meson in the $\Upsilon(4S)$ decay, the tag B. For incoherent produced B mesons this means that if a specific flavor is given to EvtGen it can return a decay of the opposite flavor. If e.g. EvtGen is given a $B^{**}+$ to decay and it decays it to $B^0\pi^+$ the decay of the B^0 might decide to flip the flavor of the decay chain to allow generating the right time integrated asymmetry. This means that after EvtGen is done the code that called EvtGen will get back a $B^{**}-$ and needs to be prepared to handle this, e.g. by applying CP to the decay chain that produced the $B^{**}+$. Also note the meaning of branching fractions in the decay table, for modes that are common to a B and its anti-particle, the $\bar{B}$ the above implies that the branching fraction listed for the B and the $\bar{B}$ are the average branching fractions.

Practically, this is implemented in the `EvtCPUtil` class, which will handle the cases of coherent and incoherently produced B mesons.

How is this different from the current implementation, V00-09-38? Not very much, there are already some models, e.g. the `SVS_NONCPEIGEN` that has the full functionality that allows for direct CP violation and will generate the right mixture of B and $\bar{B}$ tags. All CP models should convert to this more general model. For applying this to B mesons that are produced incoherent, i.e. with a definite flavor, we need to modify the `EvtCPUtil::otherB()` method to allow the change of the flavor.

10.5.2 Mixing

In the $\Upsilon(4S)$ system mixing is handled by the decay of the $\Upsilon(4S)$ meson via models such as `VSS_BMIX`. This implementation seems sufficient at this point. Note that we are not considering the case of the $\Upsilon(5S)$ at this point.

For incoherently produced neutral B mesons, B^0 or B_s, we assume that the flavor is tagged and mixing will be generated by adding a decay like $B^0 \to \bar{B}^0$ where the mixed particle have zero lifetime. Models that simulate CP violation, or in general models that handle decays common to the B and the $\bar{B}$ decay will remove any mixing, this is done in `EvtCPUtil`. (Models that do not make any assumption about the flavor of the B should probably be using a 'neutral B' particle instead of a specific flavor, but at BABAR we have not implemented this as it would be very confusing at this point to a lot of analysis code that uses Monte Carlo truth.)

This means that models that simulates CP-violating decays should not need to turn of mixing, it will practically be ignored.

The mixing of incoherently produced B mesons is not done in a model, this causes some problems as it is hard to control the mixing, e.g., turning it of or modifying the parameters. We need to invent a way to control this.

11 Semileptonic decays in EvtGen

11.1 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 11.2 we define the conventions used in the implementation of semileptonic decays. Section 11.3 describes the framework from which semileptonic form factor models can be built. An example of such a model is given in section 11.4. Finally, we list the form factor models implemented in EvtGen in section 11.5.

11.2 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

$$\begin{aligned} < S(k)|V_\mu|P(p) > &= \left\{(p+k)_\mu - \frac{m_P^2 - m_S^2}{q^2} q_\mu\right\} f_+(q^2) \\ &+ \left\{\frac{m_P^2 - m_S^2}{q^2} q_\mu\right\} f_0(q^2) \end{aligned} \tag{54}$$

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

$$\begin{aligned} < V(k)|(V-A)_\mu|P(p) > &= \epsilon^*_\mu (m_P + m_V) \mathrm{A}_1(q^2) - (p+k)_\mu (\epsilon^* p) \frac{\mathrm{A}_2(q^2)}{(m_P + m_V)} \\ &- q_\mu (\epsilon^* p) \frac{2 m_V}{q^2} (\mathrm{A}_3(q^2) - i\mathrm{A}_0(q^2)) \\ &+ \epsilon_{\mu\nu\rho\sigma} \epsilon^{*\nu} p^\rho k^\sigma \frac{2\mathrm{V}(q^2)}{m_P + m_V} \end{aligned} \tag{55}$$

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

$$\mathrm{A}_3(q^2) = \frac{m_P + m_V}{2m_V} \mathrm{A}_1(q^2) - \frac{m_P - m_V}{2m_V} \mathrm{A}_2(q^2) \tag{56}$$

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

$$\begin{aligned} < T(k)|(V-A)_\mu|P(p) > &= ih(q^2) \epsilon_{\mu\nu\lambda\rho} \epsilon^{*\nu\alpha} p_\alpha (p+k)^\lambda (p-k)^\rho - k(q^2) \epsilon^*_{\mu\nu} p^\nu \\ &+ b_+(q^2) \epsilon^*_{\alpha\beta} p^\alpha p^\beta (p+k)_\mu - b_-(q^2) \epsilon^*_{\alpha\beta} p^\alpha p^\beta (p-k)_\mu \end{aligned} \tag{57}$$

11.3 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, pseudoscalar to vector, 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 11.2. An example of how to use this framework is presented in section 11.4.

11.4 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 effictive 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:

$$A_1 = \frac{h_{A_1}(w)}{R^*}\left(1 - \frac{q^2}{(M_B + M_{D^*})^2}\right) \tag{58}$$

$$V = R_1 \frac{h_{A_1}(w)}{R^*} \tag{59}$$

$$A_2 = R_2 \frac{h_{A_1}(w)}{R^*} \tag{60}$$

where

$$h_{A_1}(w) = h_{A_1}(1)\Big(1 - \rho^2_{A_1}(w-1)\Big), \tag{61}$$

$$R^* = \frac{2.0(M_B M_{D^*})^{1/2}}{(M_B + M_{D^*})}, \tag{62}$$

and

$$w = \frac{M_B^2 + M_{D^*}^2 - q^2}{2 M_B M_{D^*}} \tag{63}$$

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 `EvtDecayAmp`, 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 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

11.5 Available models

There are 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.

11.5.1 EvtSLPole

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

$$F = \frac{F(0)}{(1.0 + a\frac{q^2}{M^2} + b\frac{q^4}{M^4})^p} \tag{64}$$

  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;
```

11.5.2 EvtISGW2

- Implementation of form factors defined from Scora and Isgur [7].

- 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.

11.5.3 EvtISGW

- Implementation of form factors defined from Isgur, Scora, Grinstein, and Wise [8].
- Leptons available: e and μ only.
- Arguements: None
- In DECAY.DEC

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

11.5.4 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.

12 Lineshape determination (new 11/2002)

12.1 Introduction

This note summarizes the proposed lineshape treatment for SP5 Monte Carlo production in `EvtGen`. Some significant changes have been made since SP4 given input from users during the past year. The generater now works in several steps:

1. The complete decay chain is determined according to the branching fractions given in `EvtGen/DECAY.DEC` and the user decay file.
2. The mass of each particle in the decay chain is determined, as described below.
3. The kinematics of each particle are determined according to the decay model specified.

As these are separate steps, the algorithm for the determination of masses is not as general is would be possible if the mass determination and kinematic determination were combined. This would entail a more significant code rewrite, which we were not able to undertake.

12.2 Lineshapes

12.2.1 What is the default?

The short answer is that it depends on the decay. For a particle P, the algorithm is as follows:

1. Something like 30% of B mesons are decayed using JETSET. If P is produced by JETSET, then its lineshape is determined by JETSET. JETSET uses a nonrelativistic Breit-Wigner.
2. If P or any of its daughters has a spin other than 0, 1/2, 1, 3/2, or 2, a nonrelativistic Breit Wigner is used.
3. If the decay $P \to D_1D_2$ proceeds by a higher partial wave than D wave, a nonrelativistic Breit Wigner is used. (in practice, this case appears to never occur in our DECAY.DEC)
4. $G \to PP_1$ and $P \to D_1D_2$ (ie, P is produced and decayed in a two body decay): A relativistic Breit-Wigner is used. The corresponding amplitude is

$$A(m) = \frac{(p_{G\to PP_1}^{L_{G\to PP_1}})(p_{P\to D_1D_2}^{L_{P\to D_1D_2}})}{m^2 - {m_0}^2 + im\Gamma(m)}(\frac{B(p_{G\to PP_1})}{B(p'_{G\to PP_1})})(\frac{B(p_{P\to D_1D_2})}{B(p'_{P\to D_1D_2})}) \tag{65}$$

   where m_0 is the nominal particle mass as specified in `PDT/pdt.table` and p is the daughter momentum in the rest frame of its parent (for the specified decay chain). p'

is the same, but for the nominal mass of P. $\Gamma(m)$ is

$$\Gamma(m) = \left(\frac{p}{p'}\right)^{2L_{P\to D_1D_2}+1} \left(\frac{m_0}{m}\right) \Gamma_0 \left[\frac{B(p)}{B(p')}\right]. \tag{66}$$

Here Γ_0 is the nominal particle width has specified in `PDT/pdt.table`. p and p' are the daughter momenta in the rest frame of P, if P has mass m or m_0, respectively. L is the angular momentum of the decay specified in its subscript, which is assumed to be the lowest allowed given the parent and daughter spins. $B(p)$ is the Blatt-Weisskopf form factor, which depends on L:

$$\begin{aligned} B(p) &= 1 \\ B(p) &= \frac{1}{\sqrt{1+(Rp)^2}} \\ B(p) &= \frac{1}{\sqrt{1+\frac{(Rp)^2}{3}+\frac{(Rp)^4}{9}}} \end{aligned} \tag{67}$$

for $L_{P\to D_1D_2}$ equal to 0, 1, or 2. R is 3 GeV^{-1}.

5. $G \to PP_1P_2X$ and $P \to D_1D_2$ (ie, P has two daughters): Very similar to the above case, except the birth momentum factor and form factor are removed:

$$A(m) = \frac{(p_{P\to D_1D_2}^{L_{P\to D_1D_2}})}{m^2 - {m_0}^2 + im\Gamma(m)} \left(\frac{B(p_{P\to D_1D_2})}{B(p'_{P\to D_1D_2})}\right) \tag{68}$$

6. $P \to P_1P_2P_3X$ (ie, P decays to more than two daughters): A nonrelativitic Breit-Wigner is used.

12.2.2 Mass cutoffs, etc

Cutoffs are listed in `PDT/pdt.table`. The cutoff given represents the maximum Δ allowed from the nominal mass, but only on the low side. That is, the lowest allowed mass is $m-\Delta$. The highest allowed mass is $m+15\Gamma$. In the case that the cutoff is not given, the lowest mass is $\max(0, m-15\Gamma)$.

12.2.3 What can be changed in user decay files?

We have added some flexibility in how lineshapes can be generated for signal Monte Carlo. We hope that this will simplify systematic studies. The various options are listed below. As with other features of user decay files, be sure to test your changes with `GeneratorsQA` before submitted your request for a Monte Carlo sample.

- Lowest generated mass

```
ChangeMassMin rho0 0.7
```

- Highest generated mass

  ```
  ChangeMassMax rho0 0.9
  ```

- Presence of the birth momentum factor and form factor (${p_P}^{L_{G\to PP_1}}$ and corresponding form factor)

  ```
  IncludeBirthFactor rho0 no
  ```

- Presence of the decay momenum factor and form factor (${p_D}^{L_{P\to D_1D_2}}$ and corresponding form factor)

  ```
  IncludeDecayFactor rho0 no
  ```

- Blatt-Weisskopf factor (in GeV^{-1})

  ```
  BlattWeisskopf rho 3.0
  ```

- Lineshape used. There are a few alternative lineshapes available:

  ```
  LSNONRELBW rho0
  LSFLAT rho0
  ```

  Note that LSFLAT should be used together with the ChangeMassMin and ChangeMassMax options.

- Mass and width

  ```
  Particle rho0 0.8 0.2
  ```

Some of these are illustrated in Figures 6 to 13 for the ρ mass in the decay $B \to \rho\pi$ and $\rho \to \pi\pi$.

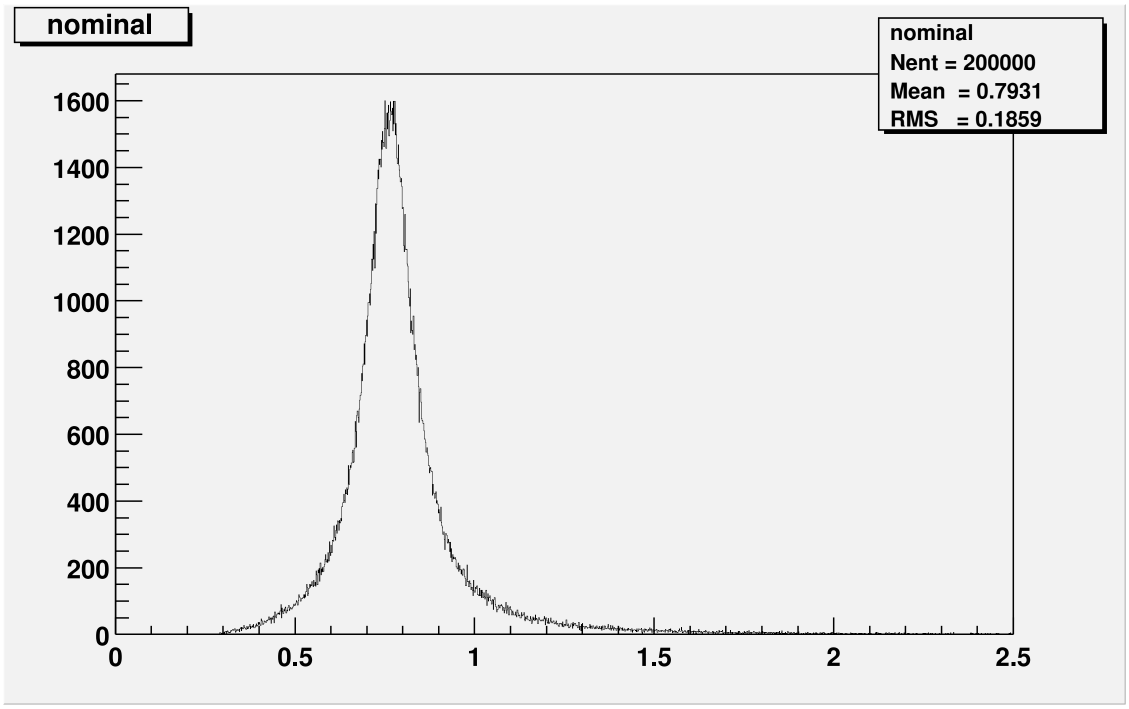

Figure 6: The nominal ρ mass distribution in $B \to \rho\pi$ and $\rho \to \pi\pi$.

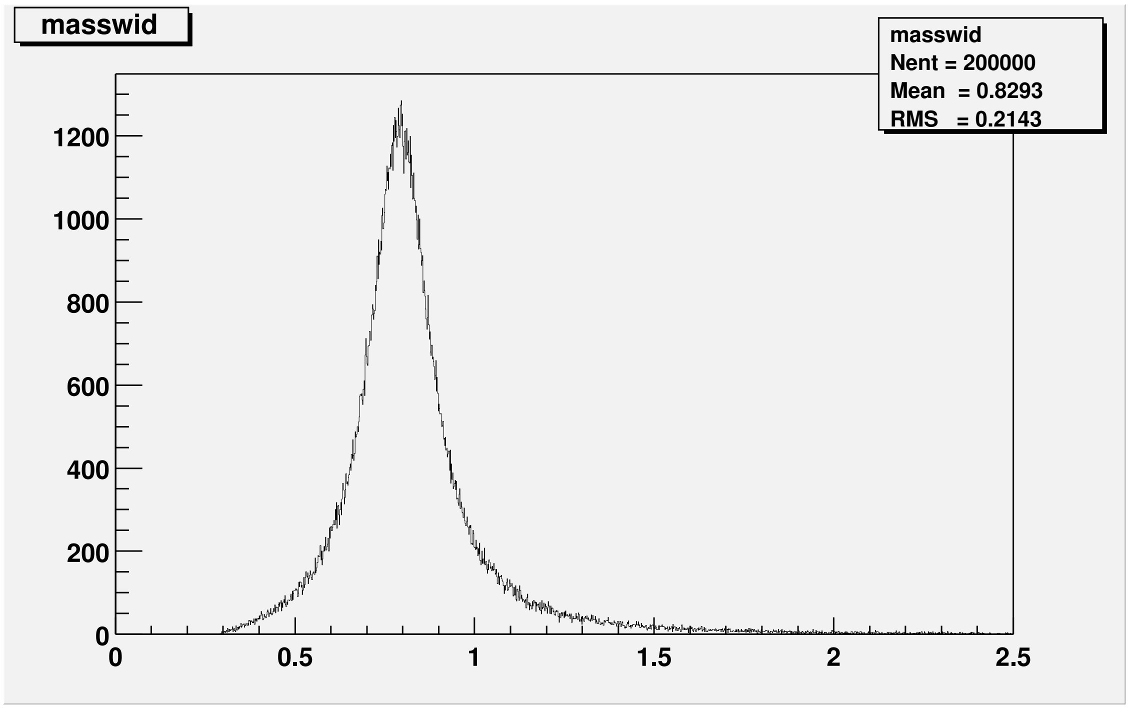

Figure 7: The ρ mass distribution in $B \to \rho\pi$ and $\rho \to \pi\pi$ having included `Particle rho0 0.8 0.2` in the user decay table.

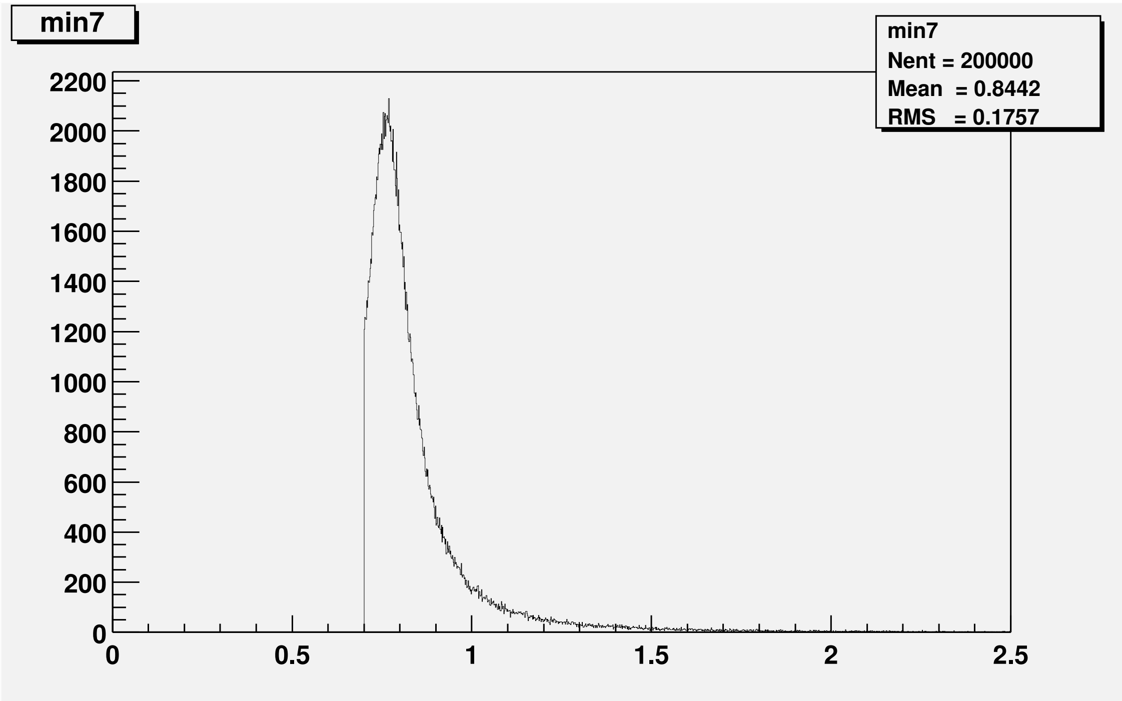

Figure 8: The ρ mass distribution in $B \to \rho\pi$ and $\rho \to \pi\pi$ having included `ChangeMassMin rho0 0.7` in the user decay table.

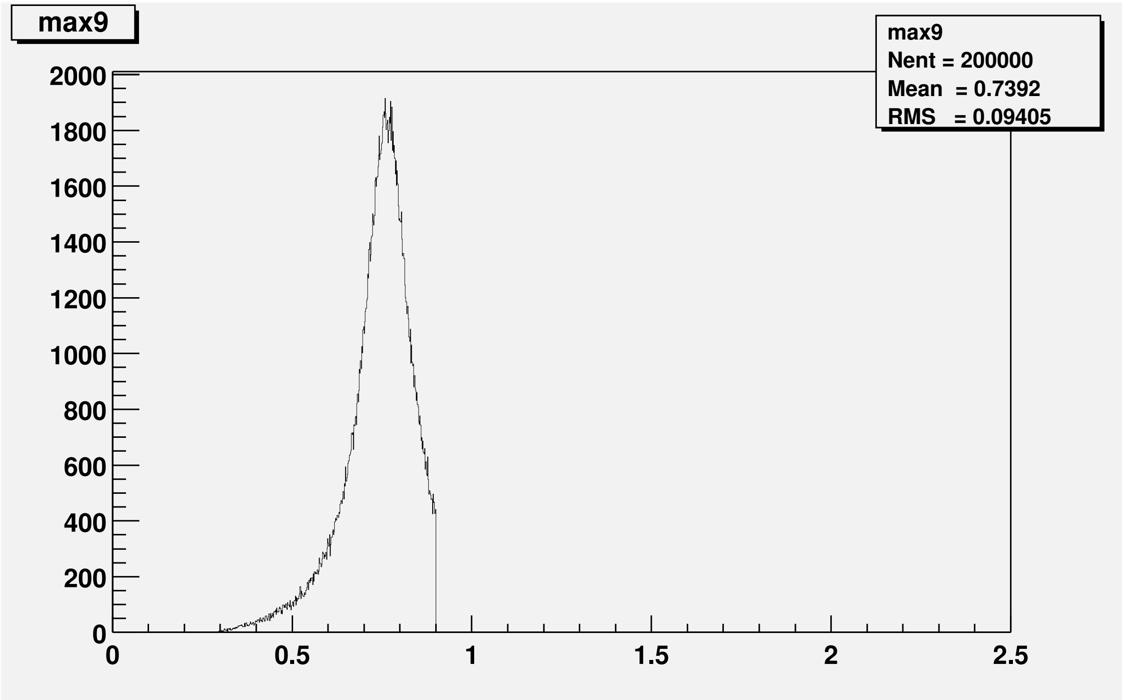

Figure 9: The ρ mass distribution in $B \to \rho\pi$ and $\rho \to \pi\pi$ having included `ChangeMassMax 0.9` in the user decay table.

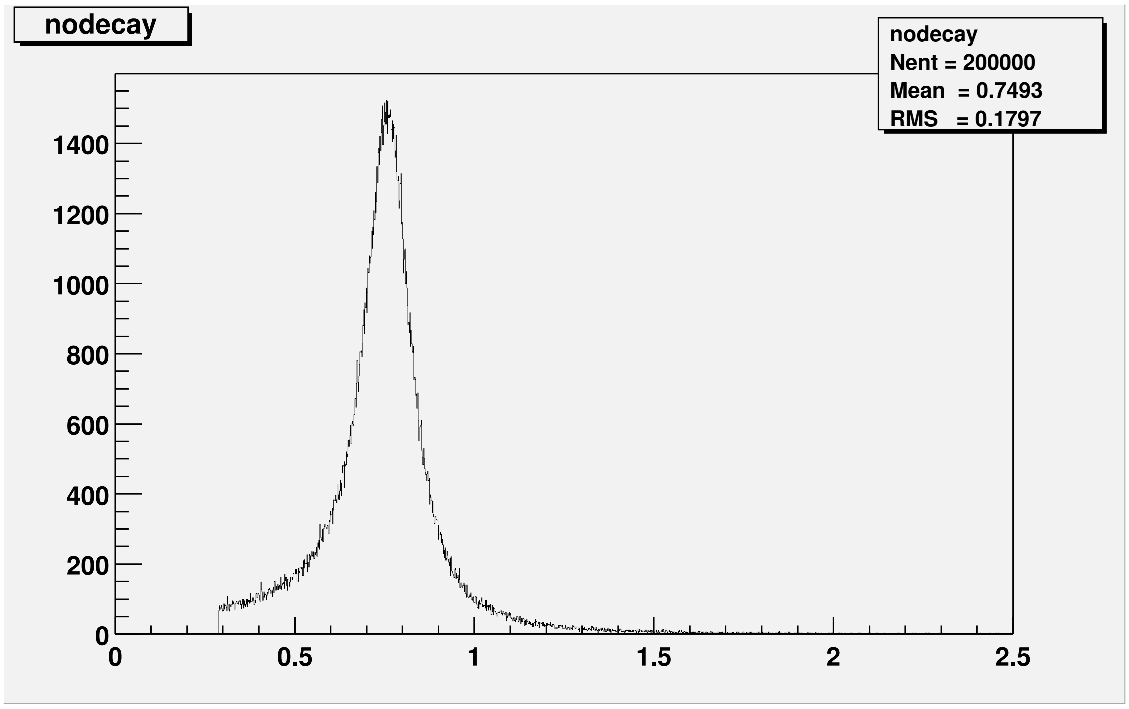

Figure 10: The ρ mass distribution in $B \to \rho\pi$, $\rho \to \pi\pi$ using `IncludeDecayFactor rho0 no` in the user decay table.

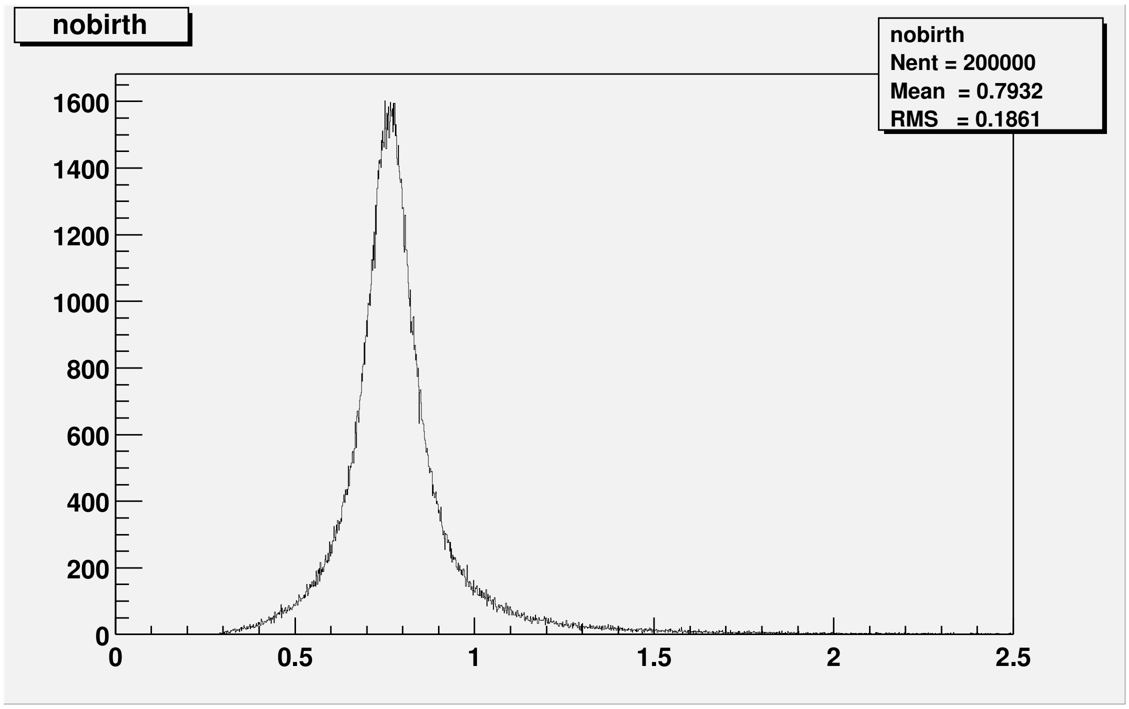

Figure 11: The ρ mass distribution in $B \to \rho\pi$, $\rho \to \pi\pi$ using `IncludeBirthFactor rho0 no` in the user decay table.

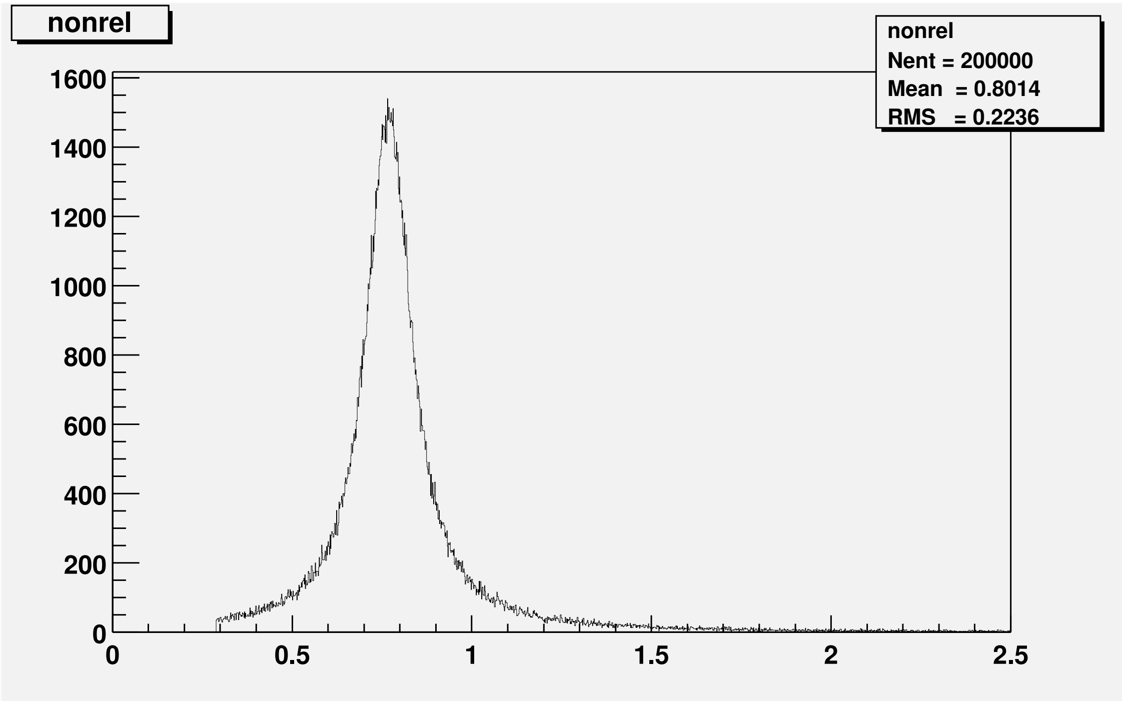

Figure 12: The ρ mass distribution in $B \to \rho\pi$ and $\rho \to \pi\pi$ having included `LSNONRELBW rho0` in the user decay table.

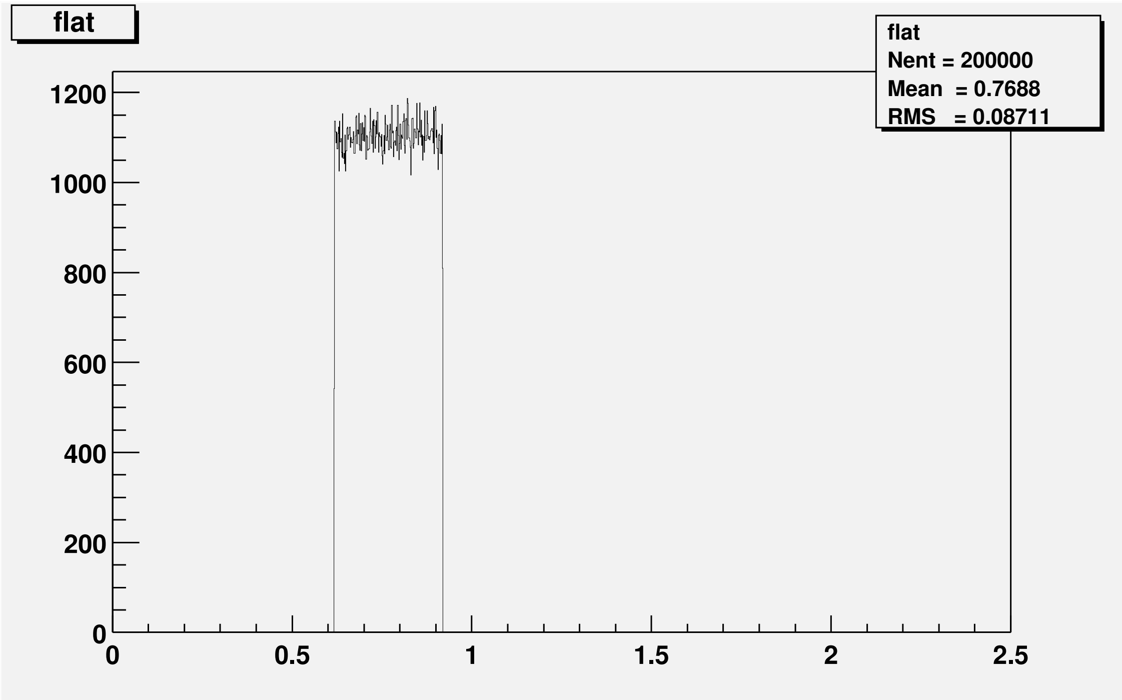

Figure 13: The ρ mass distribution in $B \to \rho\pi$ and $\rho \to \pi\pi$ having included `LSFLAT rho` in the user decay table.

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 of 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 [9]. 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 [10].

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 [11].

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` $|A_2|$ φ_{A_2} $|\overline{A}_2|$ $\varphi_{\overline{A}_2}$ $|A_0|$ φ_{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:

$$A(B^+ \to \pi^+\pi^0) \equiv A^{+0} = 3\,A_2 \tag{69}$$

$$A(B^0 \to \pi^+\pi^-) \equiv \sqrt{\frac{1}{2}}\,A^{+-} = A_2 \,-\, A_0 \tag{70}$$

$$A(B^0 \to \pi^0\pi^0) \equiv A^{00} = 2\,A_2 \,+\, A_0, \tag{71}$$

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:

$$A(B^+ \to \pi^0 K^+) \equiv A^{0+} = U - W \tag{72}$$

$$A(B^+ \to \pi^+ K^0) \equiv \sqrt{\frac{1}{2}} A^{+0} = V + W \tag{73}$$

$$A(B^0 \to \pi^- K^+) \equiv \sqrt{\frac{1}{2}} A^{-+} = V - W \tag{74}$$

$$A(B^0 \to \pi^0 K^0) \equiv A^{00} = U + W, \tag{75}$$

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 BTOXSGAMMA

Author: Francesca Di Lodovico, Jane Tinslay, Mark Ian Williams
Usage:
`BrFr P1 P2 BTOXSGAMMA model` or
Usage:
`BrFr P1 P2 BTOXSGAMMA  model F` m_B m_b μ λ_1 δ `z (number of intervals to compute` α_s`) (number of intervals tocompute the hadronic mass)`

Explanation:
This model is for two-body non-resonant $B \to X_s\gamma$ decays where strange hadrons, X_s, are

generated with a linewidth given by the mass spectrum predicted by either Ali and Greub [12] or Kagan and Neubert [13] model, according to the first parameter in the datacards given after the model is chosen. In case the Ali and Greub model is chosen, a parameterisation of the mass spectrum predicted for given inputs is used. The input parameters where based on PDG 2000 values plus a b quark Fermi momentum of 265 MeV for a spectator quark mass of 150 MeV, which was taken from CLEO fits of the semileptonic B momentum spectrum. In case the Kagan and Neubert model is used, the input parameters can be given as an input in the datacards. The are: F = Fermi momentum model (1 = exponential shape function, 2 = gaussian shape function, 3 = roman shape function), m_B, m_b, μ, λ_1, δ, z, number of intervals to compute α_s, number of intervals to compute the hadronic mass. Moreover, as a possible option, no input parameters can be given after the Kagan and Neubert model is chosen, and in this case default input parameters are chosen (F = 1, $m_B = 5.27885$ GeV/c^2, $m_b = 4.80$ GeV/c^2, $\mu = 4.80$ GeV/c^2, $\lambda_1 = 0.3$, $\delta = 0.9$, z = 0.084, number of intervals to compute $\alpha_s = 100$, number of intervals to compute the hadronic mass = 80). A a cut–off on the hadronic mass at 1.1 GeV/c^2 is applied in this case according to [13].

The maximum mass value for all X_s is 4.5 GeV/c^2 and the minimum mass value is at the Kπ threshold for X_{su} and X_{sd}, and at the KK threshold for X_{ss}. JETSET is required to decay the resulting X_s into hadrons via phase-space production from the available quarks. The decay of X_s needs to be switched on in the decay file using JETSET switches.

Example:

The example shows how to generate $B^0 \to X_{sd}\gamma$ for the Ali and Greub Model.

```
#   Xsd meson  (sbar-d system, introduced for b->s gamma decays)

# Set Xsd so it can decay:
JetSetPar MDCY(455,1)=1
# Set decay table entry pt for Xsd:
JetSetPar MDCY(455,2)=1154
# Number of decay channels for Xsd:
JetSetPar MDCY(455,3)=1
# Switch on Xsd decay
JetSetPar MDME(1154,1)=1
# Phase space decays into hadrons from available quarks
JetSetPar MDME(1154,2)=11
# Xsd decays into two quarks a d and an anti-s
JetSetPar KFDP(1154,1)=-3
JetSetPar KFDP(1154,2)=1

Decay B0
1.0000 Xsd   gamma   BTOXSGAMMA 1;
Enddecay
```

Notes:
P1 should always be reserved for the X_s particle and P2 should always be a gamma. Also, this model requires Jst74 V00-00-11 or higher to work.

A.10 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 [14].
- $D^+ \to \overline{K}^0\pi^+\pi^0$ 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 [15].
- $D^0 \to \overline{K}^0\pi^+\pi^-$ 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 [14].
- $D^0 \to K^-\pi^+\pi^0$ 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 [14].

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 [15], 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.11 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 [16], partly due to errors in the paper and because the D^* had to be removed from the $D\pi$ non-resonant.

A.12 HELAMP

Author: Ryd
Usage:
`BrFr M D1 D2 HELAMP Amplitudes;`

Explanation:
This model allows simulation of any two body decay by specifying the helicity amplitudes for the final state particles. The helicity amplitudes are complex numbers specified as pairs of magnitude and phase. The amplitudes are ordered, starting by the highest allowed helicity for the first particle. For a fixed helicity of the first particle the amplitudes are then specified starting with the highest allowed helicity of the second particle. This means that the helicities $H_{\lambda_1\lambda_2}$ are ordered first according the the value of λ_1 and then the value of λ_2.

Example:
Decay of $B^0->D^*\rho$,

```
Decay B+
1.000   anti-D*0  rho+          HELAMP 0.228 0.95 0.283 1.13 0.932 0;
Enddecay
```

Notes:
The amplitudes are taken from ICHEP 98-852. This model has been tested on many special cases, but further testing is needed.

A.13 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 [17]

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 [17] by the CLEO collaboration.

A.14 HQET2

Author: Ishikawa
Usage:
```
BrFr M L N HQET2 RHO2 R1 R2;
```

Explanation:
Model for the $D^*\ell\nu$ decay of B mesons according to the dispersive relation [18]. The daughters are in the order: D^*, lepton and last the neutrino. 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.

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

```
Decay B0
1.000 D*-  e+ nu_e             HQET2 1.35 1.3 0.8;
Enddecay
```

Notes:
The values in the example above comes from the measurement in [19] by the Belle collaboration.

A.15 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 [8]. 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 11 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.16 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 [7]. 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 11 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.17 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.

Notes:
As discussed in Appendix E the `JETSET` model can not be used to decay a particle that is an alias. This is bacause JetSet does not allow for more than one decay table per particle.

A.18 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.19 KLL3P

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

Explanation:
Implementation for the process $B \to K l^+ l^-$, as the previous model the form-factors are calculated in the framework of three-point QCD sum-rules [20].

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.20 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 [21].

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 [21] 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.21 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 [20].

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.22 LNUGAMMA

Author:edward
Usage:
`BrFr L NU GAMMA LNUGAMMA PMC R M_B FAFVZERO;`

Explanation:
Calculation of the tree-level matrix element for the process $B^+ \to l^+\nu_l\gamma$ [22].

Example:

The example shows how to generate $B^+ \to l^+\nu_l\gamma$.

```
Decay B0
1.0000 e+  nu_e  gamma  LNUGAMMA 0.35 3.0 5.0 0;
Enddecay
```

Notes:
See the citation given above for more detail.

Arg(0) is the photon mass cutoff in GeV, Arg(1) is R in GeV^{-1}, Arg(2) is m_b in GeV, and Arg(3) is set to 0 if the user wants $|f_a/f_v| = 1$, and set to 1 if the user wants $f_a/f_v = 0$. Arg(3) is optional, defaulting to 0.

A.23 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
```

A.24 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.25 PARTWAVE

Author: Ryd
Usage:
`BrFr M D1 D2 PARTWAVE Amplitudes;`

Explanation:
This model is similar to the `HELAMP` model in that it allows any two-body decay specified by the partial wave amplitudes. This model translates the partial wave amplitudes to helicity amplitudes using the Jacob Wick transformation. The partial wave amplitudes are complex numbers, specified as a magnitude and a phase. The amplitudes M_{LS} are sorted on the highest value of L and then on the highest value of S.

Example:

Decay of $B^0 - > D^* \rho$ in this example would be in pure P-wave.

```
Decay B+
1.000   anti-D*0  rho+           PARTWAVE 0.0 0.0 1.0 0.0 0.0 0.0;
Enddecay
```

Notes:
This model has been tested on some special cases, but further testing is needed.

A.26 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.27 PTO3P

Author: Dvoretskii

The PTO3P model is a generic decay-file driven model for simulating decays of a scalar (typically pseudoscalar, hence the P) particle into a final state composed of three scalar particles. $B^+ \to K^+\pi^+\pi^-$ would be an example of such a decay.

It is possible to specify several channels through which the decay can proceed. The interference effects are proprely handled by the model. It is also possible to include time-dependent mixing in decays of neutral mesons.

Explanation:
For an example of a decay, see the example below. The first two parameters specify the PDF maximum (e.g. `MAXPDF 116.2`). The PDF maximum is needed to perform accept/reject during generation. Alternatively if the maximum is not known one can specify the number of points that will be sampled `SCANPDF 10000`. The PDF will be evaluated at each point. To be conservative this maximum will be increased by 20%. Typically it's a good idea to do the scan once for a large number of events. Determine the maximum and then put it explicitly in the decay file.

The other parameters are grouped into partial `AMPLITUDE` specifications and `COEFFICIENT` specifications. Compelex coefficients can be in cartesian or polar coordinates. The keywords are: `CARTESIAN` for cartesian coordinates and `POLAR_RAD` and `POLAR_DEG` for polar coordinates.

Partial amplitudes can be either `PHASESPACE` or `RESONANCE`. For amplitudes describing intermediate resonances one should specify which two particles form the resonance (`AB`, `BC`, `CA`), and the parameters of the resonance - spin, mass and width. The resonance parameters can be specified, either as three numbers or as the particle name. In the latter case the parameters will be taken from the evt.pdl file. Finally, it's possible to get the spin from the evt.pdl file and override the mass and the width of the particle. Examples:

```
RESONANCE BC 1 0.77 0.15
RESONANCE BC rho+
RESONANCE BC rho+ 0.77 0.15
```

`ANGULAR AB` declares between which two particles the helicity angle will be evaluated. (The rest frame was specified previously, e.g. `RESONANCE BC`. This disambiguates the sign of the amplitude.

`TYPE` specifies the type of the propagator. Choose between non-relativistic Breit-Wigner (`NBW`), relativistic Zemach expression (`RBW_ZEMACH`), Kuehn-Santamaria propagator (`RBW_KUEHN`) and relativistic propagator used e.g. in `CLEO hep-ex/0011065` (`RWB_CLEO`).

Finally it's possible to supply Blatt-Weisskopf form factors at the production (birth) vertex of the resonance and its decay vertex.

```
DVFF BLATTWEISSKOPF 3.0
BVFF BLATTWEISSKOPF 1.0
```

For decays with mixing, e.g. $B^0 \to \pi^+\pi^-\pi^0$ first specify partial amplitudes for $B^0 \to \pi^+\pi^-\pi^0$, then stick in keyword `CONJUGATE` followed by mixing parameters (currently dm). Then specify partial amplitudes for $\bar{B}^0 \to \pi^+\pi^-\pi^0$ in the usual way.

For very narrow resonances the generation efficiency may be very low. In that case one should use pole compensation. In PTO3P pole-compensation is automatically turned on for all resonances. The pole-compensator PDF is created by the same factory that creates the amplitude. EvtDalitzBwPdf is used for that purpose. If you would like to switch off pole-compensation you'll need to edit EvtPto3PAmpFactory.cc, there is no way to control it via the decay file at this point.

Example:

The example below shows the decay $D^+ \to \bar{K}^0\pi^+\pi^0$ including the ρ^+ and $\bar{K}^{*0}$ resonances.

```
Decay D+

1.0 anti-K0 pi+ pi0 PTO3P
        MAXPDF          75.0
        #SCANPDF 10000
        #gives 73.5

        # Non-resonant
        AMPLITUDE       PHASESPACE
        COEFFICIENT     POLAR_RAD       0.9522  -1.8565

        # rho+ (770)
        AMPLITUDE       RESONANCE       BC      rho+            0.7699  0.1512
                        ANGULAR         AC
                        TYPE            RBW_CLEO
                        DVFF            BLATTWEISSKOPF  25.38
        COEFFICIENT     POLAR_RAD       0.389   0.0

        # anti-K*0 (770)
        AMPLITUDE       RESONANCE       AC      anti-K*0        0.89159 0.0498
                        ANGULAR         BC
                        TYPE            RBW_CLEO
                        DVFF            BLATTWEISSKOPF  10.15
        COEFFICIENT     POLAR_RAD       0.194   0.7191
```

```
         ;
Enddecay
```

A.28 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.

The last two and four arguments need not be specified. If the last two are omitted the ϕ range is from 0 to 2π and if the last four arguments are omitted the $\cos\theta$ range is from -1 to $+1$.

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
```

or simply

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

A.29 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+  nu_mu                          SLN;
Enddecay
```

A.30 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 11.5.1.

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
```

A.31 SSD_CP

Author: Ryd
Usage:
`BrFr S D SSD_CP dm dgog |q/p| arg(q/p) |A_f| argA_f |barA_f| argbarA_f`
`|A_barf| argA_barf |barA_barf| argbarA_barf |z| arg(z);`

Explanation:
This model simulates the decay of a B meson to a scalar and one other particle of arbitrary (integer) spin. An example of using this model is $B \to J/\psi K_S$

```
Decay B0
1.000 J/psi K0S    SSD_CD dm dgog |qop| arg(qop)
                          |Af| arg(Af) |Abarf| arg(Abarf)
                          |Afbar| arg(Afbar) |Abarfbar| arg(Abarfbar)
                          |z| arg(z);
Enddecay
```

where `dm` is the mass difference of the two mass eigenstates, `dgog` is $2y$, $y \equiv (\Gamma_H - \Gamma_L)/(\Gamma_H + \Gamma_L)$. `qop` is q/p where $|B_{L,H}\rangle = p|B^0\rangle \pm q|\bar{B}^0\rangle$. `Af` and `Abarf` are the amplitudes for the decay of a B^0 and a $\bar{B}^0$ respectively to the final state f. The set of amplitudes, `Afbar` and `Abarfbar` corresponds to the decay to the CP conjugate final state. These amplitudes are optional and are by default $A_{\bar{f}} = \bar{A}_f^*$ and $\bar{A}_{\bar{f}} = A_f^*$, consistent with CPT for a common final state of the B^0 and $\bar{B}^0$. However, in modes such as $B \to D^*\pi$ it is usefull to be able to specify these amplitudes separately.

The example below shows the decays $B \to J/\psi K_S$ and $B \to J/\psi K_L$

```
Define dm 0.472e12
Define minusTwoBeta -0.85
Decay B0
0.5000 J/psi K0S   SSD_CD dm 0.0 1.0 minusTwoBeta 1.0 0.0 -1.0 0.0;
0.5000 J/psi K0L   SSD_CD dm 0.0 1.0 minusTwoBeta 1.0 0.0  1.0 0.0;
Enddecay
```

Note that the sign of the amplitude for the $\bar{B}^0$ decay have the oposite sign for the K_S as this final state is odd under parity.

To generate the final state $\pi^+\pi^-$.

```
Define dm 0.472e12
Define minusTwoBeta -0.85
Define gamma 1.0
Decay B0
1.0000 pi+ pi-   SSD_CD dm 0.0 1.0 minusTwoBeta 1.0 gamma 1.0 -gamma;
Enddecay
```

These examples have used $|q/p| = 1$ and $\Delta\Gamma = 0$. An example with non-trivial values for these parameters would be $B_s \to J/\psi\eta$

```
Define dms 14e12
DEfine dgog 0.1
Decay B_s0
1.0000 J/psi eta  SSD_CD dms dgog 1.0 0.0 1.0 0.0 1.0 0.0;
Enddecay
```

This model can also be used for final states that are not CP eigenstates, such as $B^0 \to D^{*+}\pi^-$ and $B^0 \to D^{*-}\pi^+$. We can generate these decays using

```
Define dms 14e12
Define minusTwoBeta -0.85
Define gamma 1.0
Decay B0
1.0000 D*+ pi-  SSD_CD dm 0.0 1.0 minusTwoBeta 1.0 0.0 0.3 gamma;
Enddecay
```

Where the Cabibbo-suppressed decay has a relative strong phase γ with respect to the Cabibbo-favored decay.

Notes:
For more details about the treatment of CP violating decays see Section 10.

A.32 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 10.

A.33 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:

$$f = \frac{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right)}{|\overline{A}_f|^2 \left(1 + |\overline{r}_f|^2 + \frac{(1-|\overline{r}_f|^2)}{1+x_d^2}\right) + |A_f|^2 \left(1 + |r_f|^2 + \frac{(1-|r_f|^2)}{1+x_d^2}\right)} \tag{76}$$

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

$$r_f = e^{2i\,\phi_M}\,\frac{\overline{A}_f}{A_f},\ \overline{r}_f = \frac{1}{r_f} \tag{77}$$

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

$$A_f \equiv A(B^0 \to \pi^+\pi^-) = A_t\,e^{i\phi_t} + A_p\,e^{i\phi_p}\,e^{i\delta} \tag{78}$$

with

$$\overline{A}_f \equiv A(\overline{B}^0 \to \pi^+\pi^-) = A_t\,e^{-i\phi_t} + A_p\,e^{-i\phi_p}\,e^{i\delta} \tag{79}$$

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 10.

A.34 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
```

A.35 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 10.

A.36 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.37 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.38 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 10.

A.39 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:

$$A(B^+ \to V^+S^0) \equiv \sqrt{2}A^{+0} = T^{+0} + 2P_1 \tag{80}$$

$$A(B^+ \to V^0S^+) \equiv \sqrt{2}A^{0+} = T^{0+} - 2P_1 \tag{81}$$

$$A(B^0 \to V^+S^-) \equiv A^{+-} = T^{+-} + P_1 + P_0 \tag{82}$$

$$A(B^0 \to V^-S^+) \equiv A^{-+} = T^{-+} - P_1 + P_0 \tag{83}$$

$$A(B^0 \to V^0S^0) \equiv 2A^{00} = T^{0+} + T^{+0} - T^{-+} - T^{+-} - 2P_0 \tag{84}$$

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 10.

A.40 SVS_NONCPEIGEN

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

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. Otherwise, set it to 0.5 to get the physical situation. This model automatically generates the correct number of B^0 and $\bar{B}^0$ tags, depending on the specified amplitudes.

Note that the last four parameters are optional. If they are not specified, then they are evaluated from the following relations between the complex amplitudes:

$$\begin{aligned} A_{\overline{f}} &= \overline{A}_f \\ \overline{A}_{\overline{f}} &= A_f \end{aligned} \tag{85}$$

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. Note that the last 4 parameters could have been omitted, since they agree with Eq. (85)

```
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
```

For $B \to D*\pm\pi^{\mp}$, use the CKM phase betaPlusHalfGamma.

Note: Temporarily, this model only works for B0, not anti-B0. This will be fixed later.

Notes:
For more details about the treatment of CP violating decays see Section 10.

A.41 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. [23] 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. [23]. 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 10. 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.42 SVV_CPLH

Author: Ryd
Usage:
```
BrFr V1 V2 SVV_CPLH 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 including different lifetimes for the different mass eigenstates, the Light and Heavy state. This model is particularly intended for decays like $B_s \to J/\psi\phi$. The first argument is the relevant CKM angle in radians. The second argument is the $B_s - \bar{B}_s$ mass difference in s^{-1} ($> 1.8 \times 10^{12}$). The width difference is not an input parameter to the model. It is determined via the definition of B_s0L and B_s0H in the evt.pdl. The next argument is called η in Ref. [23] 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. [23]. This model then uses these amplitudes together with the time evolution of the B_s to generate the time dependent angular distributions.

Example:

This example decays the B_s meson to ϕK^{*0}

```
Decay B_s0
1.000   J/psi  phi   SVV_CPLH  0.4 3.0e12 2.0 1 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 10. This code is not well tested at all. Please be aware that there can be serious mistakes in this model!

A.43 SVS_CPLH

Author: Ryd
Usage:
```
BrFr V S SVS_CPLH dm dGoG |q/p| arg(q/p) |Af| arg(Af) |Abarf| arg(Abarf);
```

Explanation:
Decay of a neutral B meson to a scalar and a vector CP eigenstate, e.g. $B^0 \to J/\psi K_S$. The first argument is the $B^0 - \bar{B}^0$ mass difference. The second argument in $\Delta\Gamma/\Gamma$. The third and fourth argument is the magnitude and phase of q/p, and the last four arguments are the magnitude and phases of the amplitude for B^0 and $\bar{B}^0$ to decay to the final state f.

Example:
This example decays the B^0 meson to $J/\psi K_S$

```
Decay B0
1.000   J/psi  K_S0   SVS_CPLH  0.472e12 0.1  1.0  0.7  1.0 0.0 1.0 0.0;
Enddecay
```

A.44 SVV_NONCPEIGEN

Author: Kurup
Usage:
BrFr V1 V2 SVV_NONCPEIGEN
dm beta gamma $|A_{+f}|$ argA_{+f} $|A_{0f}|$ argA_{0f} $|A_{-f}|$ argA_{-f}
$|\overline{A}_{+f}|$ arg$\overline{A}_{+f}$ $|\overline{A}_{0f}|$ arg$\overline{A}_{0f}$ $|\overline{A}_{-f}|$ arg$\overline{A}_{-f}$
$|A_{+\overline{f}}|$ arg$A_{+\overline{f}}$ $|A_{0\overline{f}}|$ arg$A_{0\overline{f}}$ $|A_{-\overline{f}}|$ arg$A_{-\overline{f}}$ [optional]
$|\overline{A}_{+\overline{f}}|$ arg$\overline{A}_{+\overline{f}}$ $|\overline{A}_{0\overline{f}}|$ arg$\overline{A}_{0\overline{f}}$ $|\overline{A}_{-\overline{f}}|$ arg$\overline{A}_{-\overline{f}}$; [optional]

Explanation:
This model is based on the SVS_NONCPEIGEN model and allows the generation of CP violation in scalar $\to$ vector + vector decays, where the final state is not a CP-eigenstate. The first argument is the $B^0 - \bar{B}^0$ mass difference. The second argument is the angle beta. The third argument is the angle relevant to the decay mode being generated. In the example below it is gamma (in fact, it's enough to specify 2 beta + gamma, perhaps in the next round of fixes). The next 24 arguments are the magnitudes and phases of the amplitudes for the four types of decay, A_f, $\overline{A}_f$, $A_{\overline{f}}$ and $\overline{A}_{\overline{f}}$, which are split into the three different helicity states $+$, 0 and $-$. Depending on the specified amplitudes, the final state will be charge conjugated and the correct number of B^0 and $\bar{B}^0$ tags are generated.

Note that the last 12 parameters are optional. If they are not specified, then they are evaluated according to the following relation between the complex amplitudes (with $i = +, 0, -$):

$$A_{i\overline{f}} = \overline{A}_{if}$$

$$\overline{A}_{i\overline{f}} = A_{if} \tag{86}$$

Example:

This example will generate $B \to D^{*\pm}\rho^{\mp}$ final states with the appropriate number of B^0 and $\bar{B}^0$ tags. The helicity amplitude parameters chosen for the first line are those measured by CLEO. The amplitudes on the second line are identical, but suppressed by a factor of 100. The last two lines were omitted, so that Eq (86) takes effect:

```
Alias MYB B0
Decay Upsilon(4S)
1.00  MYB anti-B0
Enddecay
Decay MYB
1.000 D*- rho+    SVV_NONCPEIGEN dm beta gamma
                                 0.152   1.47 0.936   0 0.317   0.19
                                 0.00152 1.47 0.00936 0 0.00317 0.19;
Enddecay
```

Note: Temporarily, this model only works for B0, not anti-B0. This will be fixed later.

A.45 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.46 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 $\tau \to e\nu_e\bar{\nu}_tau$

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

A.47 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.48 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.49 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.50 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.51 VECTORISR

Author: Zallo,Ryd
Usage:
`BrFr VECTOR GAMMA VECTORISR CSFWMN CSBKMN;`

Explanation:
Generates the interaction, $e^+e^- \to V\gamma$ where V is a vector meson according to [24]. 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                    VECTORISR 0.878 0.95;
Enddecay
```

Notes:
This model produces an unpolarized vector meson.

A.52 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.53 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.54 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.55 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. Use the new VSS_BMIX model to generate mixing in the correct proportions using a single decay channel. See Section 10 for more details about how mixing is implemented and how it works with CP violation.

A.56 VSS_BMIX

Author: Kirkby
Usage:
`BrFr B1 B2 VSS_BMIX dm;`

Explanation:
Decays a C=-1 vector particle into two scalar particles using $B^0\bar{B}^0$-like coherent mixing. The two possible daughter particles must be charge conjugates and have the same lifetime. Their mass difference is supplied as an argument to the model, in units of $\hbar$/s. While the mass difference is a required arguement, $\Delta\Gamma/\Gamma$ and $|q/p|$ can be supplied as optional arguements, with defaults of 0 and 1 respectively. The examples below illustrate how this model accomadates aliased daughters.

Example:
The example shows how to generate $\Upsilon(4S) \to B^0\bar{B}^0$ decays with coherent mixing (but without CP violating effects).

```
Define dm 0.474e12
Decay Upsilon(4S)
  1.0 B0  anti-B0  VSS_BMIX dm;
Enddecay
```

to include a non-zero $\Delta\Gamma/\Gamma$:

```
Define dm 0.474e12
Define dgog 0.5
Decay Upsilon(4S)
  1.0 B0  anti-B0  VSS_BMIX dm dgog;
Enddecay
```

and to specify $|q/p|$

```
Define dm 0.474e12
Define dgog 0.5
Define qoverp 1.2
Decay Upsilon(4S)
  1.0 B0  anti-B0  VSS_BMIX dm dgog qoverp;
Enddecay
```

Finally, aliased particles can be generated using this model

```
Define dm 0.474e12
alias myB0 B0
alias myanti-B0 anti-B0
Decay Upsilon(4S)
  1.0 B0  anti-B0 myB0 myanti-B0 VSS_BMIX dm;
Enddecay
```

generates either `B0 myanti-B0`, `anti-B0 myanti-B0`, `myB0 anti-B0`, or `myB0 B0`.

Notes:
This model is similar to the VSS_MIX model, but it eliminates the need to manually specify the fractions of mixed and un-mixed events through branching fractions. This approach has the effect that the resulting mixing distributions are necessarily self consistent, which is not true for the VSS_MIX model when using the wrong branching fractions. See Section 10 for more details about how mixing is implemented and how it works with CP violation.

A.57 VVPIPI

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

Explanation:
This decay model was constructed for the decay $\psi' \to J/\psi\pi^+\pi^-$ but should work for any $V \to V'\pi\pi$ decay in which the approximation that the $\pi\pi$ system can be treated as one particle which combined with the V' meson is dominated by S-wave. The amplitude for the mass of the $\pi\pi$ sstem is given by $A \propto (m^2_{\pi\pi} - 4m^2_\pi)$.

Example:

```
Decay psi(2S)
1.000 J/psi pi+ pi-          VVPIPI;
Enddecay
```

A.58 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.59 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 [25]. 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 11 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 [3] 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.

The functionality of aliases in EvtGen and the `JETSET` model don't work well together. The implementation of aliases in EvtGen creates a new decay table for the aliased particle. However, JetSet don't have the functionality to allow two, or more, different decay tables for the same particle. Hence, EvtGen will not allow you to decay a particle that is an alias using the `JETSET` model.

F Helicity amplitudes

This section will deal with some details related to helicity amplitudes and their relations to partial wave amplitudes. In particular the sign conventions are described and the relations between the Jackson [26] and the Jackob-Wick [27] conventions for the helicity formalism is explained.

This section is not meant as a complete guide to the use of helicity amplitudes. There are several references that gives a good introduction to the use of helicity amplitudes for describing the dynamics of particle decays. Richman [28] gives a pedagogical introduction following the conventions of Jacob and Wick [27]. Jackson [26] uses a slightly different choice of conventions. One of the main purposes of this section is to describe these conventions and establish a map between the two conventions. Both of these conventions are used 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. To make sure that there is no ambiguity about definitions and conventions the next section provides definitions for the terminology used.

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 right handed it is left handed – there is no other alternative. In the discussion below all coordinate systems considered are right handed.

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

$$R(\alpha, \beta, \gamma) \equiv R_{z'}(\gamma) R_{y'}(\beta) R_z(\alpha) \tag{87}$$

where this means that a rotation by α is first performed around the z-axis. Then a rotation by β around the y'-axis is done. Where the y'-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 in terms of rotations around a fixed coordinate system,

$$R(\alpha, \beta, \gamma) = R_z(\alpha) R_y(\beta) R_z(\gamma). \tag{88}$$

See e.g. Sakurai [29] page 171-174 for illustrations of the rotations.

The $D^J_{m,m'}(\alpha, \beta, \gamma)$ functions are defined by

$$D^J_{m,m'}(\alpha, \beta, \gamma) \equiv \langle Jm | R(\alpha, \beta, \gamma) | Jm' \rangle. \tag{89}$$

Using $d^J_{m,m'}(\beta) \equiv \langle Jm | e^{-i\beta J_y} | Jm' \rangle$\$ we can write

$$D^J_{m,m'}(\alpha, \beta, \gamma) = e^{-im\alpha} d^J_{m,m'}(\beta) e^{-im'\gamma}. \tag{90}$$

F.2 Plane wave states

The state $\Psi_{p\lambda}$ denotes a state with momentum p along the z-axis with helicity λ. This state is obtained by applying a boost $L(p)$ along the z-axis to the state $|J = s \quad m = \lambda\rangle$, where $|Jm\rangle$ are the canonical angular momentum states. The total angular momentum, s, is suppressed in the notation below.

Following Jacob and Wick [27] we define the states $\chi_{p\lambda}$ which have momentum p along the negative z-direction

$$\chi_{p\lambda} = (-1)^{s-\lambda} e^{-i\pi J_y} \Psi_{p\lambda}. \tag{91}$$

First we will verify that this state has the properties that we expect;

$$\begin{aligned}
J_z \chi_{p\lambda} &= J_z (-1)^{s-\lambda} e^{-i\pi J_y} \Psi_{p\lambda} \\
&= (-1)^{s-\lambda} e^{-i\pi J_y} e^{i\pi J_y} J_z e^{-i\pi J_y} \Psi_{p\lambda} \\
&= (-1) s - \lambda e^{-i\pi J_y} (-1) J_z \Psi_{p\lambda} \\
&= -\lambda \chi_{p\lambda}
\end{aligned} \tag{92}$$

shows that the eigen value of J_z is $-\lambda$ as expected. Further, we look at the application of the lowering operator, $J_- = J_x - iJ_y$, on the state $\chi_{p\lambda}$

$$\begin{aligned}
J_- \chi_{p\lambda} &= (-1)^{s-\lambda} (J_x - iJ_y) e^{-i\pi J_y} \Psi_{p\lambda} \\
&= (-1)^{s-\lambda} e^{-i\pi J_y} e^{i\pi J_y} (J_x - iJ_y) e^{-i\pi J_y} \Psi_{p\lambda} \\
&= (-1)^{s-\lambda} e^{-i\pi J_y} (-J_x - iJ_y) \Psi_{p\lambda} \\
&= -(-1)^{s-\lambda} e^{-i\pi J_y} J_+ \Psi_{p\lambda} \\
&= -(-1)^{s-\lambda} e^{-i\pi J_y} \sqrt{(s-\lambda)(s+\lambda+1)} \Psi_{p\lambda+1} \\
&= \sqrt{(s-\lambda)(s+\lambda+1)} (-1)^{s-(\lambda+1)} e^{-i\pi J_y} \Psi_{p\lambda+1}
\end{aligned} \tag{93}$$

which shows that the application of J_- on $\Psi_{p\lambda}$ behaves as expected for a particle with helicity λ along the negative z-direction. Note in particular that the factor $(-1)^\lambda$ is important to ensure the right phase when applying the lowering operators. In fact, the properties that we have demonstrated above almost shows the following property. If $p = 0$, i.e. the particle is at rest, then

$$\chi_{0\lambda} = \Psi_{0-\lambda}. \tag{94}$$

The properties of the lowering and raising operators and J_z that we have shown above shows that the states $\chi_{0\lambda}$ and $\Psi_{0-\lambda}$ differs at most by a phase and that this phase is independent of λ. However, it is easiest to show Eq. 94 using the Wigner d-functions

$$\chi_{0\lambda} = (-1)^{s-\lambda} e^{-i\pi J_y} \Psi_{0\lambda} = (-1)^{s-\lambda} \sum_{\lambda'} d^s_{\lambda'\lambda}(\pi) \Psi_{0\lambda'} = \Psi_{0-\lambda} \tag{95}$$

since

$$d^s_{mm'}(\pi) = (-1)^{s-\lambda} \delta_{m,-m'}. \tag{96}$$

An alternative way to define the states $\chi_{p\lambda}$ is by

$$\chi_{p\lambda} = (-i)^{2s} e^{-i\pi J_x} \Psi_{p\lambda}. \tag{97}$$

we find that

$$\chi_{0\lambda} = (-i)^{2s} e^{-i\pi J_x} \Psi_{0\lambda} = (-i)^{2s} \sum_{\lambda'} \tilde{d}^s_{\lambda'\lambda}(\pi) \Psi_{0\lambda'} = \Psi_{0-\lambda} \tag{98}$$

where

$$\tilde{d}^s_{m'm}(\theta) = \langle jm'|e^{-i\theta J_x}|jm\rangle. \tag{99}$$

The $\tilde{d}$ function is similar to the standard Wigner d function except that it refers to expectation values for rotations around the x-axis instead of the y-axis. An explicit formula for the $\tilde{d}$ function is given by

$$\begin{aligned} \tilde{d}^j_{m'm}(\theta) &= \sum \frac{\sqrt{(j+m)!(j-m)!(j+m')!(j-m')!}}{(m+m'+k)!(j-m'-k)!k!(j-m-k)!} \times \\ & \left(\cos\frac{\theta}{2}\right)^{m'+m+2k} \left(-i\sin\frac{\theta}{2}\right)^{2j-2k-m'-m} \end{aligned} \tag{100}$$

from which it is easy to show that

$$\tilde{d}^j_{m'm}(\pi) = (-1)^{2j} \delta_{m',-m} \tag{101}$$

It is straight forward to show that the application of the raising and lowering operators as well as the operators J_z on the state defined by Eq. 97 is what is expected. Note in particular that there is no longer any need for phase factors to get the right phase on the different helicity states. This is particularly useful because it allows us to simply interpret the states $\chi_{p\lambda}$ in the rest frame of the particle with the momentum p along the negative z direction. That is, the state $\chi_{p\lambda}$ is equal to $|J = s \quad m = \lambda\rangle$ in a coordinate system that has been rotated by π around the x axis. This is very important because it tells us how to construct the coordinates systems in sequential decays when applying the helicity formalism, this will be discussed further below.

We now define the two particle plane wave states when particle A is along the positive z-direction by

$$|p\lambda_A\lambda_B\rangle = \Psi_{p\lambda_A}\chi_{p\lambda_B}. \tag{102}$$

Applying J_z to this state we find

$$J_z|p\lambda_A\lambda_B\rangle = (\lambda_A - \lambda_B)|p\lambda_A\lambda_B\rangle \tag{103}$$

States where the relative momentum, p, is not along the z-axis are obtained by rotating the state $|p\lambda_A\lambda_B\rangle$. Let θ and ϕ denote the polar coordinates for particle A, then the we define the state $|p\theta\phi\lambda_A\lambda_B\rangle$ by

$$|p\theta\phi\lambda_A\lambda_B\rangle = R(\alpha,\beta,\gamma)|p\lambda_A\lambda_B\rangle \tag{104}$$

where $R(\alpha, \beta, \gamma)$ is an Euler rotation. Here there is a choice of conventions. Jackson [26] takes $\alpha = \phi$, $\beta = \theta$, and $\gamma = 0$ while Jacob and Wick [27] uses $\alpha = \phi$, $\beta = \theta$, and $\gamma = -\phi$. The choice of the angle γ is arbitrary but has to be used consistently.

We have now defined the state $|p\theta\phi\lambda_A\lambda_B\rangle$ and explored its properties in some detail. This state is used as the final state in the two body decays in the helicity formalism.

Before we are ready to use these states we need to construct a set of states that are labeled by p, λ_A, and λ_B and have definite values of J and m, i.e., are eigenstates of J^2 and J_z. That such states exists is obvious since p, λ_A, and λ_B are invariant under rotations. We denote these states by $|pJM\lambda_A\lambda_B\rangle$. The relation between these states and the plane wave states created above is given by

$$|pJM\lambda_A\lambda_B\rangle = \sqrt{\frac{2J+1}{4\pi}} \int_{d\Omega} D^{*J}_{M\lambda_A-\lambda_B}(\phi, \theta, -\phi)|p\theta\phi\lambda_A\lambda_B\rangle. \tag{105}$$

Note that the states $|pJM\lambda_A\lambda_B\rangle$ are independent of the choice of Euler angles.

F.3 Helicity amplitudes

We will now consider the decay $C \to A + B$. The initial particle, C, is assumed to be in the state $|JM\rangle$ and the final state, $A+B$, is $|pJM\lambda_A\lambda_B\rangle$. We will assume that the interaction U that causes this transition is invariant under rotation, but is otherwise arbitrary. We which to evaluate the matrix element

$$M = \langle pJM\lambda_A\lambda_B\rangle|U|JM\rangle. \tag{106}$$

To do this we insert the identity written as

$$I = \sum_{p'J'M'\lambda'_A\lambda'_B} |p'J'M'\lambda'_A\lambda'_B\rangle\langle p'J'M'\lambda'_A\lambda'_B| \tag{107}$$

and use Eq. 105.

$$\begin{aligned}
M &= \langle p\theta\phi\lambda_A\lambda_B|U|JM\rangle \\
&= \sum_{p'J'M'\lambda'_A\lambda'_B} \langle p\theta\phi\lambda_A\lambda_B|p'J'M'\lambda'_A\lambda'_B\rangle\langle p'J'M'\lambda'_A\lambda'_B|U|JM\rangle \\
&= \langle p\theta\phi\lambda_A\lambda_B|pJM\lambda_A\lambda_B\rangle\langle pJM\lambda_A\lambda_B|U|JM\rangle \\
&= \sqrt{\frac{2J+1}{4\pi}}\langle p\theta\phi\lambda_A\lambda_B| \int_{d\Omega'} D^{*J}_{M,\lambda_A-\lambda_B}(\phi', \theta', -\phi')|p\theta'\phi'\lambda_A\lambda_B\rangle\langle pJM\lambda_A\lambda_B|U|JM\rangle \\
&= \sqrt{\frac{2J+1}{4\pi}} D^{*J}_{M,\lambda_A-\lambda_B}(\phi, \theta, -\phi)H_{\lambda_A\lambda_B}
\end{aligned} \tag{108}$$

where the helicity amplitudes are defined by

$$H_{\lambda_A\lambda_B} = \langle pJM\lambda_A\lambda_B|U|JM\rangle. \tag{109}$$

F.4 Helicity amplitudes and sequential decays

The previous section explained how the helicity formalism is used to calculate the amplitudes for a two body decay, $A \to B + C$. It is straight forward to now to use this in sequential decays. The only thing that requires a it of care is the construction of the coordinate systems in which the decay angles are measured.

To explain how to use the helicity formalism in sequential decays we will consider the decay chain $A \to B + C$, $B \to D + E$, and $C \to F + G$. The initial particle, A, is in the state $|J = J_A \quad m = \lambda_A\rangle$. The amplitudes for the decay $A \to B + C$ is now given by

$$A^{A\to B+C}_{\lambda_A\lambda_B\lambda_C} = \sqrt{\frac{2J_A+1}{4\pi}} D^{*J_A}_{\lambda_A,\lambda_B-\lambda_C}(\phi_B, \theta_B, -\phi_B) H^A_{\lambda_B\lambda_C} \tag{110}$$

where θ_B and ϕ_B are the polar angles of particle B in the rest frame of particle A.

Similarly, we can write the amplitude for the decay of particle B

$$A^{B\to D+E}_{\lambda_B\lambda_D\lambda_E} = \sqrt{\frac{2J_B+1}{4\pi}} D^{*J_B}_{\lambda_B,\lambda_D-\lambda_E}(\phi_D, \theta_D, -\phi_D) H^B_{\lambda_D\lambda_E}. \tag{111}$$

The coordinate system in which the angles θ_D and ϕ_D are measured is obtained by rotating the coordinate system of the parent particle, A, using the same Euler angles as was used when calculating the amplitude for the decay of particle A. This means that the coordinate system for particle B is obtained by doing the rotation $R(\phi_B, \theta_B, \phi_B)$ of the coordinate system of particle A.

In the same way we obtain the amplitude for the decay of particle C,

$$A^{C\to F+G}_{\lambda_C\lambda_F\lambda_G} = \sqrt{\frac{2J_C+1}{4\pi}} D^{*J_C}_{\lambda_C,\lambda_F-\lambda_G}(\phi_F, \theta_F, -\phi_F) H^C_{\lambda_F\lambda_G}. \tag{112}$$

As discussed above the coordinate system for the second particle, here particle C in the decay of particle A, is obtained by rotating the coordinates system of the first particle, B, by π around its x axis. I.e. the x-axis of particles B and C frames are parallel.

F.4.1 Jackson convention

In the Jackson convention the Euler rotation is taken to be $R(\phi, \theta, 0)$ where the angles (θ, ϕ) are the polar coordinates for $\vec{p}$. This is a rotation first by ϕ around the z-axis and then a rotation by θ around the new y-axis. The new y-axis is in the direction of $z \times \vec{p}$ and the new x-axis is therefore in the direction $(z \times \vec{p}) \times \vec{p}$.

This is a simple geometrical construction that allows the construction of the coordinate system used with the Jackson convention for the Euler rotations.

F.4.2 Jacob-Wick convention

The coordinate system used in the Jacob-Wick convention is obtained by performing an additional rotation of $-\phi$ around the new z-axis as obtained in the Jackson conventions.

In the sense that there is a simple geometrical construction for the coordinate system in the Jackson convention if might be argued that this convention is somewhat simpler to use.

F.5 Explicit representations of SU(2)

This section describes the explicit representations of SU(2) that are used in the generator. As usual $\hbar = 1$.

F.5.1 $J = 1/2$

$$S_n = \frac{1}{2}\gamma_0\gamma_5 \not{n}. \tag{113}$$

F.5.2 $J = 1$

We will consider the representation of spin 1.

$$J_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{bmatrix}, \quad J_y = \begin{bmatrix} 0 & 0 & i \\ 0 & 0 & 0 \\ -i & 0 & 0 \end{bmatrix}, \quad J_z = \begin{bmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{114}$$

From these explicit representations it is trivial to show that they obey the standard commutation relations, e.g.,

$$[J_x, J_y] = iJ_z \tag{115}$$

Further we find, as expected, that

$$J^2 = J_x^2 + J_y^2 + J_z^2 = 2I \tag{116}$$

such that $J^2 = j(j+1)$ which shows that this in fact is a representation of spin 1. The raising and lowering operators are as usual given by

$$J_+ = J_x + iJ_y = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & -i \\ 1 & i & 0 \end{bmatrix}, \quad J_- = J_x - iJ_y \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -i \\ -1 & i & 0 \end{bmatrix} \tag{117}$$

It is also convenient to evaluate the expression for a finite rotation

$$R_z(\theta) = e^{-i\theta J_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{118}$$

Of course we could have guessed the form of $R_z(\theta)$, but an explicit evaluation of $e^{-i\theta J_z}$ is possible through direct evaluation of the series expansion, but is more elegantly done using Cayley-Hamilton's theorem.[2]

[2] Cayley-Hamilton's theorem states that if $P_A(\lambda) = det(\lambda I - A)$ then $P_A(A) = 0$ for any symmetric matrix A. This means that if A has dimension n then A^k can be written as a linear combination of A^i for $i < n$. In particular it can be shown that $f(A) = q(A)$ where q is a polynomial of degree $< n$. The polynomial $q(A)$ is defined by

$$\frac{d^j f}{dz^j}(\lambda_k) = \frac{d^j q}{dz^j}(\lambda_k) \tag{119}$$

where λ_k are the eigen values of A, with multiplicity n_k and $j = 0..n_k - 1$.

The states

$$\epsilon_+ = \frac{1}{\sqrt{2}}\begin{bmatrix} -1 \\ -i \\ 0 \end{bmatrix}, \quad \epsilon_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \epsilon_- = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -i \\ 0 \end{bmatrix} \tag{120}$$

are easily seen to satisfy

$$J_z \epsilon_\lambda = \lambda \epsilon_\lambda \tag{121}$$

and therefore form a helicity eigenstate basis for a particle with $J = 1$ moving in the direction of the positive z-axis.

F.6 Projections of helicity amplitudes

This section shows how the helicity amplitudes are related to the invariant amplitudes. Consider the decay of a scalar to two vector particles. Let the final states of the two vector particles be denoted by $\epsilon^1_{s_1}$ and $\epsilon^2_{s_2}$ and $\vec{p}$ be the relative momentum of the two particles in the parents rest frame. The amplitude for this process is then written as

$$A_{s_1 s_2} = \epsilon^{1*}_{i s_1} \epsilon^{2*}_{j s_2} M_{ij}(\vec{p}) \tag{122}$$

where $M_{ij}(p)$ is a rank 2 tensor as a function of the available momenta in the process, i.e., a function of $\vec{p}$. The most general form of M is given by

$$M_{ij} = a\delta_{ij} + b\epsilon_{ijk}p_k + cp_i p_j. \tag{123}$$

The coefficients a, b, and c are the invariant amplitudes, we wand to relate them to helicity amplitudes. First we note that there are three invariant amplitudes, this is the same as the number of helicity amplitudes.

From Eq. 108 the amplitude is given by

$$A_{\lambda_1 \lambda_2} = \sqrt{\frac{2J+1}{4\pi}} H_{\lambda_1 \lambda_2} D^{*J}_{M \lambda_1 - \lambda_2}(\phi, \theta, -\phi). \tag{124}$$

When relating the helicity amplitudes to the invariant amplitudes it is sufficient to look at one kinematic configuration, we chose the simplest possible in which $\theta = \phi = 0$. From the definition of the D function it is obvious that $D^{*J}_{M\lambda_1-\lambda_2}(0,0,0) = \delta_{M,\lambda_1-\lambda_2}$. This gives

$$H_{\lambda_1 \lambda_2} = \sqrt{\frac{4\pi}{2J+1}} A_{\lambda_1 \lambda_2}. \tag{125}$$

This allows us to simply evaluate the helicity amplitudes from the invariant amplitudes if we chose the states, ϵ^1 and ϵ^2, to correspond to the states in the helicity formalism. For ϵ^1 we take the states given by 120

$$\epsilon^1_+ = \frac{1}{\sqrt{2}}\begin{bmatrix} -1 \\ -i \\ 0 \end{bmatrix}, \quad \epsilon^1_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \epsilon^1_- = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -i \\ 0 \end{bmatrix}. \tag{126}$$

The states for the second particle have to satisfy $\chi_{0\lambda} = \Psi_{0-\lambda}$ which gives

$$\epsilon^2_+ = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -i \\ 0 \end{bmatrix}, \quad \epsilon^2_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \epsilon^2_- = \frac{1}{\sqrt{2}}\begin{bmatrix} -1 \\ -i \\ 0 \end{bmatrix}. \tag{127}$$

Note that this is consistent with Eq. 97. Now it is straight forward to evaluate the helicity amplitudes

$$H_{++} = \sqrt{\frac{4\pi}{3}}(-a - ib), \tag{128}$$

$$H_{00} = \sqrt{\frac{4\pi}{3}}(a + c), \tag{129}$$

$$H_{--} = \sqrt{\frac{4\pi}{3}}(-a + ib). \tag{130}$$

These equations are easily inverted to give

$$a = -\frac{1}{2}\sqrt{\frac{3}{4\pi}}(H_{++} + H_{--}), \tag{131}$$

$$b = \frac{i}{2}\sqrt{\frac{3}{4\pi}}(H_{++} - H_{--}), \tag{132}$$

$$c = \sqrt{\frac{3}{4\pi}}(H_{00} + \frac{1}{2}(H_{++} + H_{--})), \tag{133}$$

$$\tag{134}$$

F.7 Jacob-Wick transformation

This section derives the relation between partial wave amplitudes and helicity amplitudes.

The amplitude for a decay, given the partial wave amplitudes M_{Ls}, is given by

$$A(\Omega, m1, m2, m3) = \sum_{L,s} C_{s_2 s_3}(s\ m_s; m_2\ m_3) C_{Ls}(s_1\ m_1; m_L\ m_s) Y_L^{m_L}(\Omega) M_{Ls}.$$

F.8 HELAMP and PARTWAVE model implementations

The two models `HELAMP` and `PARTWAVE` implements generic two body decays where the amplitudes are specified by the helicity and partial wave amplitudes respectively. The partial wave model uses the translation to helicity amplitudes as given in Sect. F.7. Henceforth, this section will focus in the description of the evalution of helicity amplitudes.

EvtGen uses a set of states, that are not the same as the helicity states. In this section it is important to distinguish between these two sets of states. The basis states used by EvtGen will be called $|n\rangle$ and the helicity states will be denoted $|\lambda, s\rangle$, or simply $|\lambda\rangle$ >.

We consider the decay of the form $A \to BC$ where the state of the initial particle is given by $|n_A\rangle$ and the two particles in the final state are labeled by $|n_B\rangle$ and $|n_C\rangle$. For convenience we label the final state $|n_B, n_C\rangle = |n_B\rangle \otimes |n_C\rangle$. Given this notation we write the amplitude that we need to implement as

$$A^{B\to BC}_{n_A,n_B,n_C} = \langle n_B, n_C|H|n_A\rangle.$$

Using the completeness of the basis states we write

$$\begin{aligned}
A^{B\to BC}_{n_A,n_B,n_C} &= \sum_{\lambda_A,\lambda_B,\lambda_C} \langle n_B, n_C|\lambda_B\rangle\langle\lambda_B|\lambda_C\rangle\langle\lambda_C|H|\lambda_A\rangle\langle\lambda_A|n_A\rangle && (135)\\
&= \sum_{\lambda_A,\lambda_B,\lambda_C} \langle n_B|\lambda_B\rangle\langle n_C|\lambda_C\rangle\langle\lambda_B|\langle\lambda_C|H|\lambda_A\rangle\langle\lambda_A|n_A\rangle && (136)\\
&= \sum_{\lambda_A,\lambda_B,\lambda_C} R^*_{n_B\lambda_B} R^*_{n_C\lambda_C}\langle\lambda_B|\langle\lambda_C|H|\lambda_A\rangle R_{\lambda_A n_A} && (137)
\end{aligned}$$

Where

$$\begin{aligned}
R_{n_A\lambda_A} &= \langle\lambda_A|n_A\rangle && (138)\\
R_{n_B\lambda_B} &= \langle\lambda_B|n_B\rangle && (139)\\
R_{n_C\lambda_C} &= \langle\lambda_C|n_C\rangle && (140)
\end{aligned}$$

G The routines `decay_angle` and `decay_angle_chi`

This section will describe the utility routines, `EvtDecayAngle`, `EvtDecayPlaneNormalAngle`, and `EvtDecayAngleChi`, that compute decay angles. The function `EvtDecayAngle` is useful for finding the decay angle, known as the helicity angle, in a decay tree that includes a two body decay. Teh routine `EvtDecayPlaneNormalAngle` calculates the angle of the decay plane normal in a three-body decay. `EvtDecayAngleChi` 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 `EvtDecayAngle`, the syntax is

```
costheta=EvtDecayAngle(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 14. 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

$$\cos\theta = \frac{(P\cdot D)M_Q^2 - (P\cdot Q)(Q\cdot D)}{\sqrt{[(P\cdot Q)^2 - M_Q^2 M_P^2][(Q\cdot D)^2 - M_Q^2 M_D^2]}}. \quad (141)$$

To call `EvtDecayPlaneNormalAngle` the syntax is

```
costheta=EvtDecayPlaneNormalAngle(P,Q,D1,D2),
```

where `P`, `Q`, `D1`, and `D2` are of type `EvtVector4R`. This routine returns the cosine of the angle θ of the normal to the decay plane. The angle calculated is that between the normal of the decay plane formed by the daughter mesons, `D1` and `D2`, in the rest frame of `Q` (the parent of `D1` and `D2`), with respect to `Q`'s flight direction in `P`'s (the parent of `Q`) rest frame. `P`, `Q`, `D1`, and `D2` are the momentum four-vectors of these particles in any frame of reference. The decay angle is computed using the (manifestly invariant) expression

$$\frac{\cos\theta = M_Q P \cdot L}{\sqrt{[(P \cdot Q)^2 - M_P^2 M_Q^2][-L^2]}} \tag{142}$$

where $L_\nu = \epsilon_{\nu\mu\alpha\beta} Q^\nu D_1^\alpha D_2^\beta$.

The routine `EvtDecayAngleChi` is used to calculate the azimuthal angel, χ, between the decay planes of a pair of two body decays. As illustrated in Figure 15, 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=EvtDecayAngleChi(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 15.

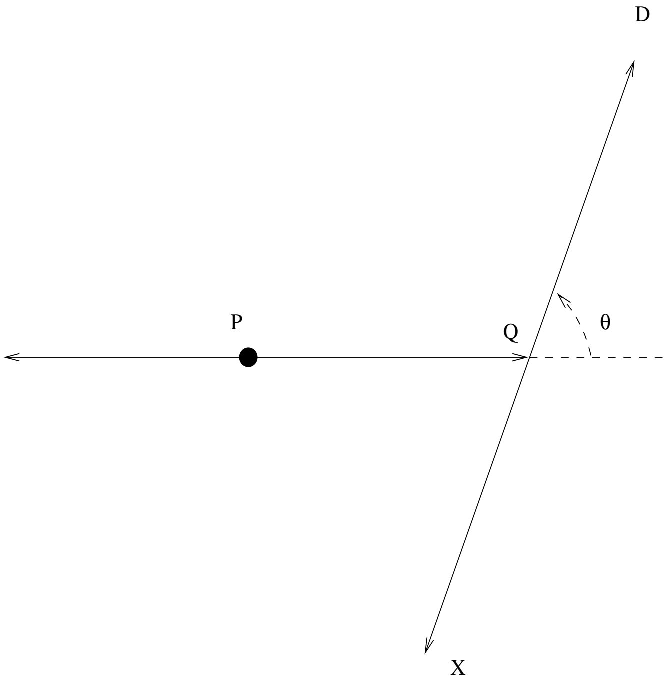

Figure 14: Definition of the decay angle.

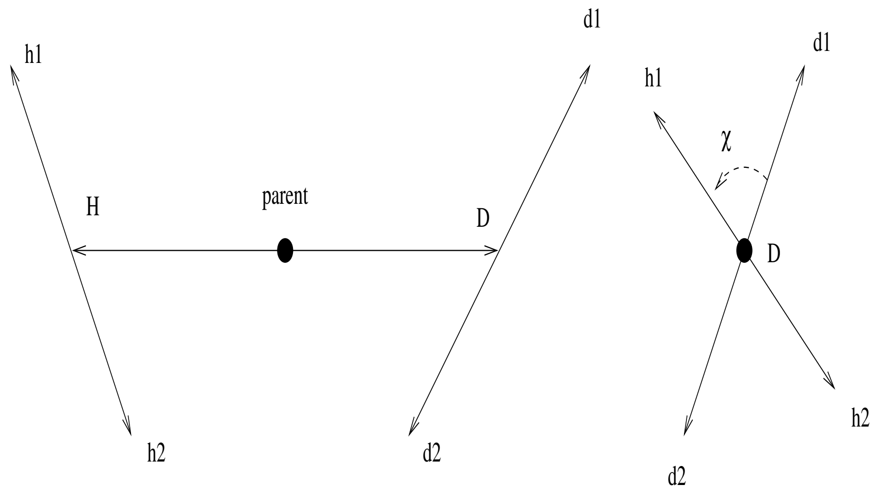

Figure 15: Definition of the χ angle.

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 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 coding 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] See http://www.lns.cornell.edu/public/CLEO/soft/QQ.

[2] T. Sjöstrand, Computer Physics Commun. **82** 74 (1994).

[3] E. Richter-Was, Phys. Lett. **B303**, 163, (1993).

[4] What can be referenced for CLHEP?

[5] http://root.cern.ch/.

[6] Gangof4.

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

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

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

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

[11] 3π section of the BaBar Book.

[12] A.Ali and C.Greub, Phys. Lett. B **361**, 146 (1995).

[13] A.Kagan and M.Neubert, Euro. Phys. Jour **7**, 5 1999.

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

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

[16] J.L. Goity and W. Roberts, Phys. Rev. **D51** 3459-3477 (1995).

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

[18] I. Caprini *et al.*, Nucl. Phys. B **530**, 153 (1998).

[19] K. Abe *et al.*, (Belle) Phys. Lett. B **526**, 247 (2002).

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

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

[22] G.P. Korchemsky, D. Pirjol, and T. Yan, *Phys. Rev.* **D61**, 114510 (2000).

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

[24] I have lost this reference...

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

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

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

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

[29]

Index