

EvtGen tutorial intro

David Lange

Lawrence Livermore Laboratory

Anders Ryd

California Institute of Technology

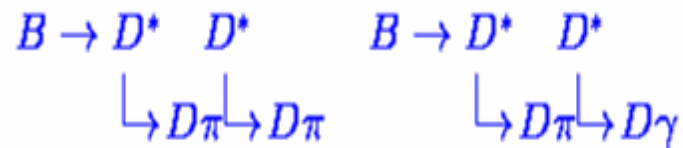
Outline

- EvtGen overview decay algorithm
- Content of evt.pdl
- Available decay models
- Decay table structure
- How to add new decay models
- Getting started with EvtGen
- B mesons from hadronic interactions
- Data MC tuning

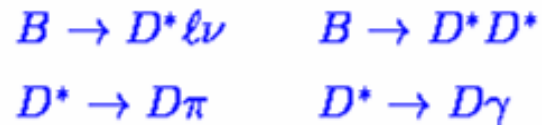
Overview and decay algorithm

Sequential decays

- Many B meson decays have interesting sequential decay chains:

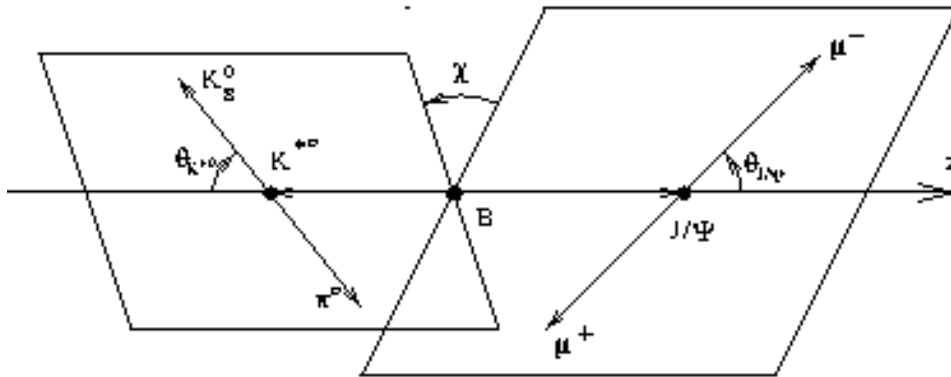
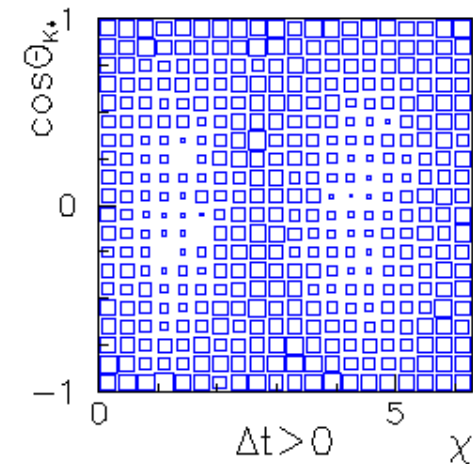
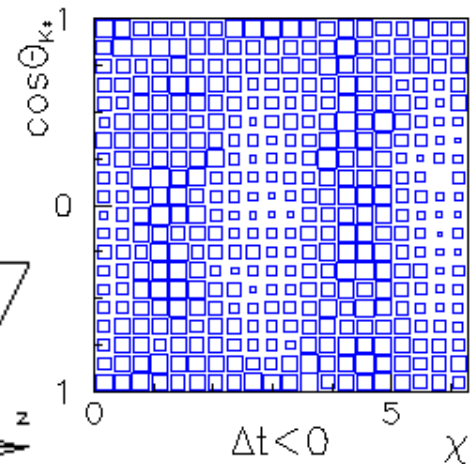
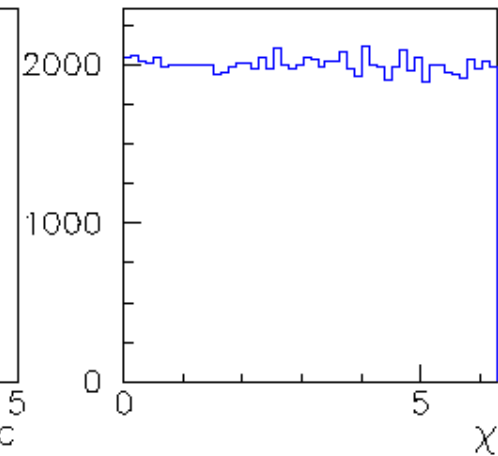
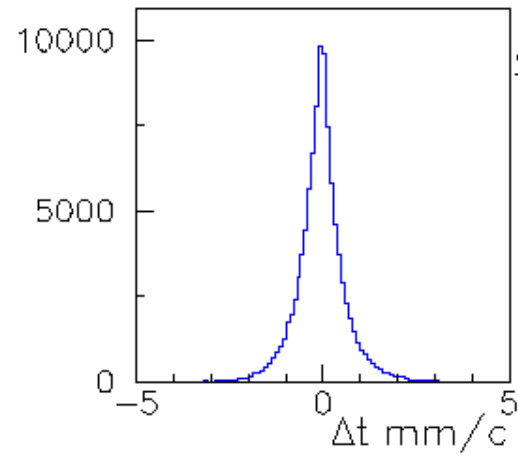


- Want to correctly simulate these decay chains while only implementing the nodes in the decay tree.

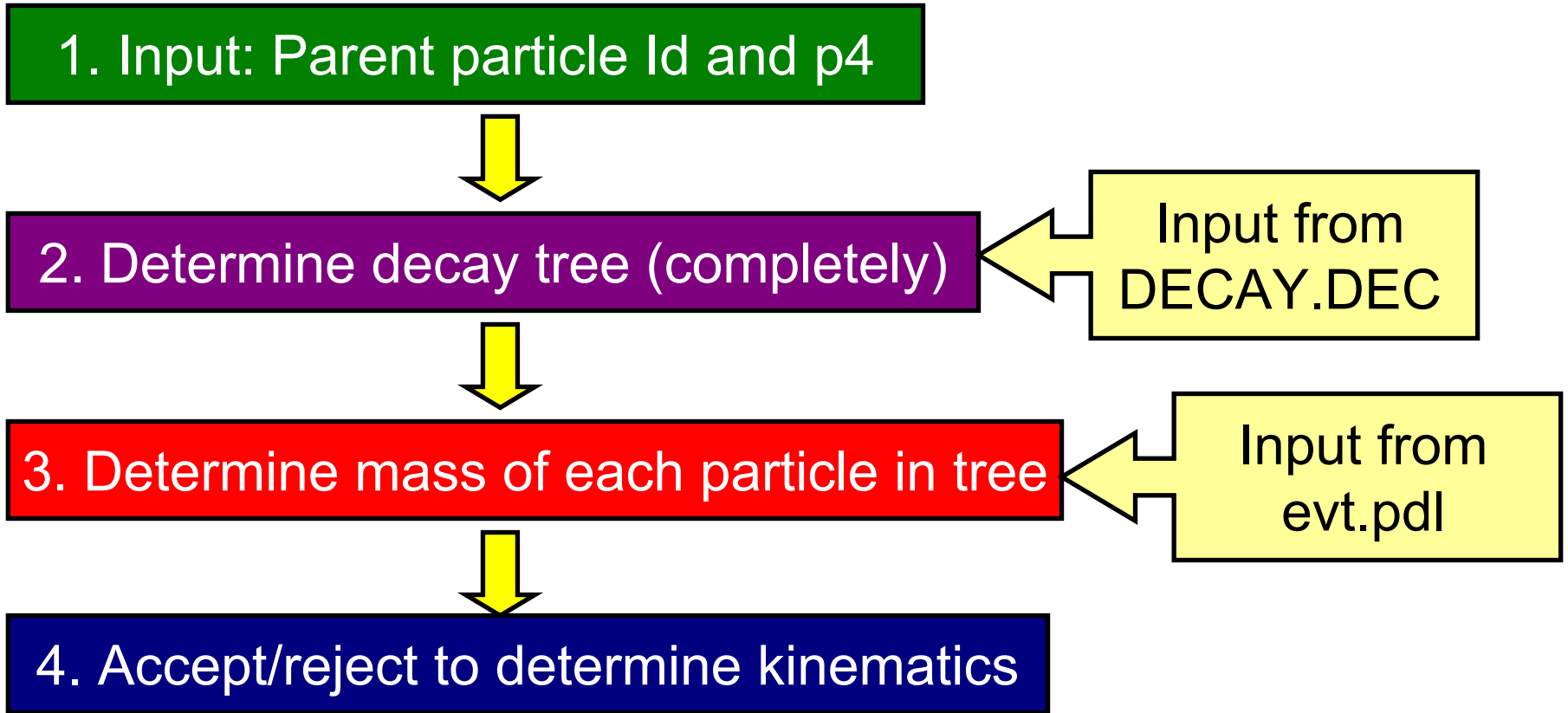


CP violating decays

- $B \rightarrow J/\psi K^{*0} \quad (K^{*0} \rightarrow K_S \pi^0)$
 - Angular correlations and time dependence



EvtGen decay algorithm



- Configuration specified by input files at run time.

Resonance decay modes and branching fractions are specified in DECAY.DEC input file

- If particle is not listed in DECAY.DEC, it is assumed to be stable.
- Given parent particle:
 - Select decay mode based on allowed decay modes for specified particle (and its mass).
 - Do not yet determine daughter masses
 - Continue down the decay chain until all particles left over are stable.
 - Some sanity checks on kinematically allowed decay sequences is included. Given broad resonances, it is in principle possible to generate sequences with no allowed set of particle masses.

After the full decay chain is determined, the mass of each particle is determined

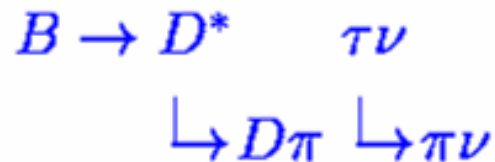
- Input from evt.pdl:
 - Nominal mass and width
 - Low mass cutoff (if any)
 - If no cutoff specified, allowed range of masses is $\pm 15\Gamma$
- Lineshapes are nominally relativistic Breit-Wigners.
 - However there are exceptions:
 - Decays to more than two daughters
 - Particles produced by Pythia (inclusive decays)
 - Where possible, birth and decay vertex factors are included (ie $B \rightarrow \rho\pi$, $\rho \rightarrow \pi\pi$)
 - More accurate lineshapes, but algorithm for lineshape depends on parent and daughter particles.

Finally, the decay kinematics are determined

- We fix the decay chain and particle masses before determining the kinematics
 - One exception is for inclusive decays from Pythia, where the daughters are determined in same step as kinematics.
 - We avoid needing to normalize decay models properly for their mass dependence.
 - This procedure not always ideal.
 - Alternative is to include resonant substructure with decay probability calculation.
 - One implemented example: Dalitz decays
 - » $D \rightarrow K\pi\pi^0$ instead of $D \rightarrow K^*\pi^0$, etc

Decay amplitudes are used instead of probabilities

- EvtGen works with amplitudes to correctly handle sequential decays:



$$d\Gamma = |A|^2 d\phi \quad A = \sum_{\lambda_{D^*} \lambda_\tau} A_{\lambda_{D^*} \lambda_\tau}^{B \rightarrow D^* \tau \nu} A_{\lambda_{D^*}}^{D^* \rightarrow D \pi} A_{\lambda_\tau}^{\tau \rightarrow \pi \nu}$$

$$A_{\lambda_{D^*} \lambda_\tau}^{B \rightarrow D^* \tau \nu} \equiv \langle \lambda_{D^*} \lambda_\tau | H | B \rangle$$

$$\sum_{\lambda_{D^*}} |\lambda_{D^*}\rangle \langle \lambda_{D^*}| = I$$

- Nodes in the decay tree are implemented as “models”. The framework of EvtGen handles the bookkeeping needed to correctly generate the full decay tree.

Selection algorithm (I)

- Generate the $B \rightarrow D^* \tau \nu$ decay

$$P = \sum_{\lambda_{D^*} \lambda_{\tau}} |A_{\lambda_{D^*} \lambda_{\tau}}^{B \rightarrow D^* \tau \nu}|^2$$

- Compare with maximum probability and accept or reject generated $B \rightarrow D^* \tau \nu$ decay.
 - Maximum probability specified in code.
 - Can instead be generated on the fly, however this leads to the output of event N depending on the random number sequence used to determine the max probability.
- Regenerate $B \rightarrow D^* \tau \nu$ decay until combination is accepted.

Selection algorithm (II)

- Average over t spin and calculate the D^* spin density matrix:

$$\rho_{\lambda_{D^*}\lambda'_{D^*}}^{D^*} = \sum_{\lambda_\tau} A_{\lambda_{D^*}\lambda_\tau}^{B \rightarrow D^* \tau \nu} (A_{\lambda'_{D^*}\lambda_\tau}^{B \rightarrow D^* \tau \nu})^*$$

- Generate the $D^* \rightarrow D\pi$ decay

$$P = \sum_{\lambda_{D^*}\lambda'_{D^*}} \rho_{\lambda_{D^*}\lambda'_{D^*}}^{D^*} A_{\lambda_{D^*}}^{D^* \rightarrow D\pi} (A_{\lambda'_{D^*}}^{D^* \rightarrow D\pi})^*$$

- Compare with maximum probability and accept or reject generated $D^* \rightarrow D\pi$ decay
- Regenerate $D^* \rightarrow D\pi$ decay until accepted. The $B \rightarrow D^* \tau \nu$ decay is **not** regenerated.

Selection algorithm (III)

- Calculate the spin density matrix for the τ

$$\rho_{\lambda_\tau \lambda'_\tau}^\tau = \sum_{\lambda_{D^*} \lambda'_{D^*}} \hat{\rho}_{\lambda_{D^*} \lambda'_{D^*}}^{D^*} A_{\lambda_{D^*} \lambda_\tau}^{B \rightarrow D^* \tau \nu} (A_{\lambda'_{D^*} \lambda'_\tau}^{B \rightarrow D^* \tau \nu})^*$$

- Where:

$$\hat{\rho}_{\lambda_{D^*} \lambda'_{D^*}}^{D^*} \equiv A_{\lambda_{D^*}}^{D^* \rightarrow D \pi} (A_{\lambda'_{D^*}}^{D^* \rightarrow D \pi})^*$$

- Generate the $\tau \rightarrow \pi \nu$ decay

$$P = \sum_{\lambda_\tau \lambda'_\tau} \rho_{\lambda_\tau \lambda'_\tau}^\tau A_{\lambda_\tau}^{\tau \rightarrow \pi \nu} (A_{\lambda'_\tau}^{\tau \rightarrow \pi \nu})^*$$

- Compare with maximum probability and accept or reject generated $\tau \rightarrow \pi \nu$ decay.
- Regenerate $\tau \rightarrow \pi \nu$ decay until accepted. The $B \rightarrow D^* l \nu$ and $D^* \rightarrow D \pi$ decays are not regenerated.

Advantages to using decay amplitudes

- Implementation of decay models is simplified by using amplitudes instead of probabilities.
- Keeping track of the spin density matrices allows us to generate each node of the decay chain independently.
 - More efficient
 - Avoids the need to determine uncountable # of maximum probabilities
- Generalizes to arbitrarily long decay chains
- Calculation of probabilities and spin density matrices are done by the framework. Models specify only the decay amplitudes.

End of overview and algorithm

evt.pdl and particle properties

evt.pdl format

Particle properties are defined in evt.pdl:

```
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 0.138570 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
```

.....

- 4th column=particle name, 5th=stdhep number, 6th=mass (GeV/c²), 7th=Width (GeV/c²), 8th=Mass cutoff, 9th=3*charge, 10th 2*spin, 11th=ct (mm), 12th Lund-KC number (for Pythia interface)

Meaning of “Mass cutoff”

- Given decays of broad resonances to other broad resonances, we find that we must cut off the mass distribution of particles to improve the robustness of our code.
- Nominal mass range:
$$m_0 - 15\Gamma < m < m_0 + 15\Gamma$$
- If the mass cutoff in evt.pdl is not 0:
$$m_0 - \text{Field\#8} < m < m_0 + 15\Gamma$$
- (Of course parent, daughter particle masses can also limit the mass range)

Notes on evt.pdl

- Interface adopted is from BABAR standard, based on MC++ (we think..)
- Quantum numbers such as C, P not included. Would in principle allow models to determine some of the information currently passed in as arguments in DECAY.DEC.

Implemented models in EvtGen

Many different models are implemented in EvtGen. They vary from highly specialized to rather generic. A rough grouping of these models into categories would be:

- Semileptonic decays
- CP violation
- Generic amplitudes
- Special matrix elements

Semileptonic decays

- HQET - Heavy Quark Effective Theory inspired form factor param.
- ISGW, ISGW2 - Quark model based prediction, Isgur, Scora et al.
- MELIKHOV - Quark model based prediction
- SLPOLE - Generic specification of form factors based on a lattice inspired parametrization.
- VUB - For generic $b \rightarrow u l \nu$ decays, uses JetSet for fragmentation.
- GOITY_ROBERTS - Decays to non resonant $D^{(*)} \pi l \nu$.

BABAR uses, HQET, ISGW2, VUB, and GOITY_ROBERTS in its simulation.

ISGW2 should support D , D_s and B_s decays as well as B decays.

CP violation in B decays

- SSD_CP - generic model for two-body decays that are common final states of the B^0 and the anti- B^0 . Includes effects of both the mass and width differences and should apply equally well to the B_s system.
- SVV_CPLH - Model for decays with two vectors in the final state, e.g. $B_s \rightarrow J/\psi \phi$.
- BTO3PI_CP, BTO4PI_CP, BTO2PI_CP_ISO, BTOKPI_CP_ISO specialized models.

Generic amplitudes

- HELAMP, PARTWAVE - generic two-body decays specified by the helicity or partial wave amplitudes.
- SLN - Decay of scalar to lepton and neutrino.
- PHSP - N-body phase space.
- SVS, STS - Scalar decay to vector (or tensor) and scalar.
- VSS, TSS - decay of vector or tensor particle to a pair of scalars.
- VLL, SLL - Decay of vector or scalar to two leptons.
- VSP_PWAVE, vector to scalar and photon, e.g.,
 $D^* \rightarrow D\gamma$

Special matrix elements

- BTOXSGAMMA - $b \rightarrow X_s \gamma$ with JetSet fragmentation.
- BTOXSELL - $b \rightarrow X_{sll}$ with JetSet fragmentation.
- D_DALITZ - 3-body D-decays with substructure.
- ETA_DALITZ - eta to 3pions with measured dalitz amplitude.
- KSTARNUUNU - $B \rightarrow K^* n \bar{u}$
- LNUGAMMA - $B \rightarrow l \nu \gamma$
- OMEGA_DALITZ - Dalitz structure in the omega- \rightarrow 3-pion decay
- PHI_DALITZ - Dalitz structure in the phi- \rightarrow 3-pion decay
- PTO3P - scalar to 3 scalars decay where you can specify intermediate resonances
- TAUHADNU - hadronic 1, 2, and 3 pion final states.
- TAULNUNU - leptonic tau decays.
- VSS_BMIX - Upsilon(4S) to $B \bar{B}$, including mixing.
- VVPIPI - decay of vector to vector and two pions, e.g. $\psi' \rightarrow \psi + \pi + \pi$.
- VECTORISR - ISR production of vector mesons:
 $e^+e^- \rightarrow V + \gamma$

DECAY.DEC and writing new decay files

Generic decay file layout

Define global parameters

Jetset/Pythia control parameters (any needed changes
from the default settings)

.....

Define decays for Particle_1

Define decays for Particle_2

....

....

Define decays for Particle_N

End

Lines starting with “#” are comments

Parameter definitions

- Use “Define” to define a keyword that can be used elsewhere in the decay file:

Define alpha 1.365

Define beta 0.39

Define dm 0.489e12

- In principle these parameters can appear anywhere in the decay file.
 - If multiply defined, second definition takes precedence only after it has been defined

Multiple parameter definitions

Define minus2Beta -0.78

Decay B0

1. J/psi K_S0 SSD_CP dm 0. 1. minus2Beta 1. 0. -1. 0.

Enddecay

Define minus2Beta -0.72

Decay B0

1. J/psi K_L0 SSD_CP dm 0. -1. minus2Beta 1. 0. -1. 0.

Enddecay

Will give J/psi K_S0 decays with $2\beta=0.78$ and J/psi K_L0 decays with $2\beta=0.72$.

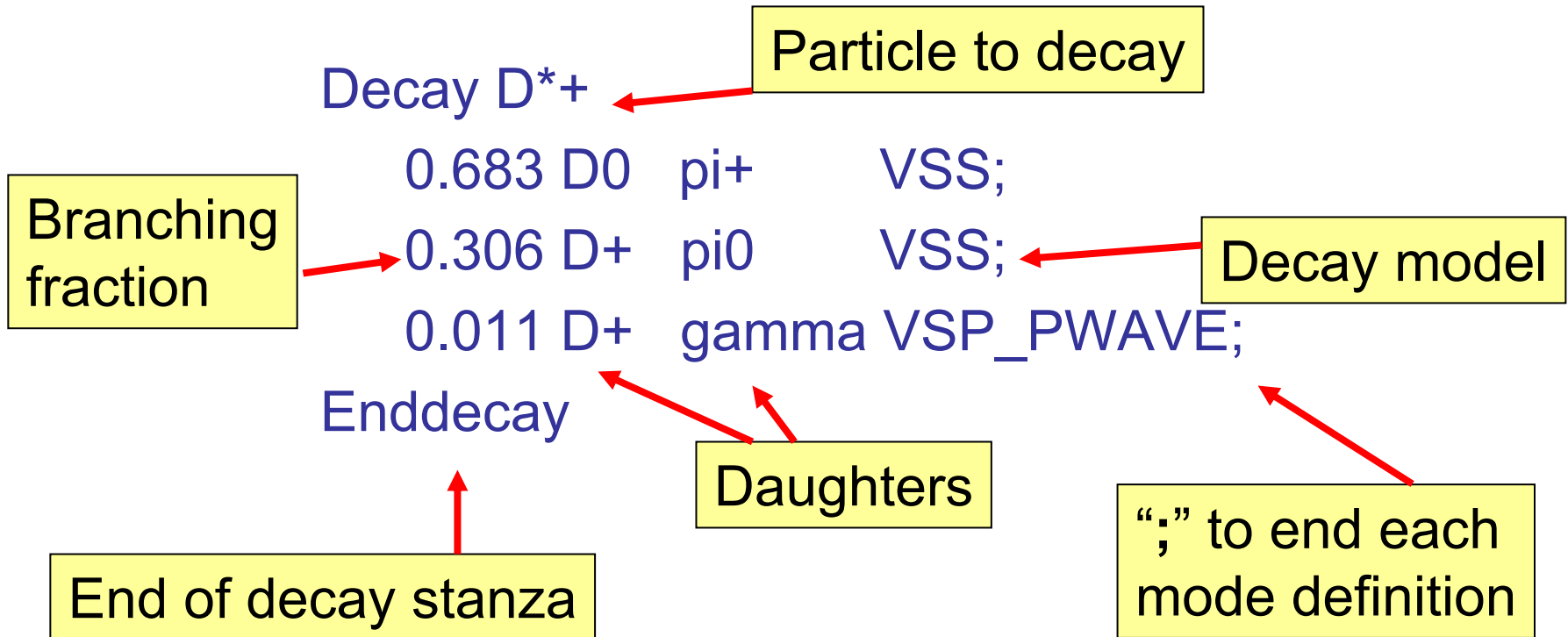
Pythia parameters

- For parameters not specified in the decay table, the default from Pythia is used.
- JetSetPar PARJ(54)=-0.040
- JetSetPar MSTJ(107)=1
- Note: no spaces allowed between parameter name, "=", and its value.

PHOTOS control

- PHOTOS can be enabled for all decays, disabled for all decays, or controlled for individual decays. Add one of the following keywords to control:
 - **yesPhotos**
 - Turns on PHOTOS for all decays
 - **noPhotos**
 - Turns off PHOTOS for all decays
 - **normalPhotos**
 - PHOTOS controlled on a decay by decay basis (to be explained in a few slides.)

Defining particle decays



Defines three decay modes of the D^{*+}

Branching fractions will be rescaled to sum to 1.0

Another example of defining decay modes

Decay anti-B0

```
0.056 D*- e+ nu_e PHOTOS HQET 0.92 1.18 0.72;  
0.021 D- e+ nu_e PHOTOS ISGW2;
```

Enddecay

Turn on PHOTOS for this decay
(unless otherwise controlled)

Decay model
arguments

In general each line in the decay file contains:

```
<BF> <Daughter1> ... <DaughterN> <PHOTOS>  
<Model_to_use> <Model_arguments>;
```

<PHOTOS> is optional. <Model_arguments> needed as required by specified model.

How to define a particle as stable

Decay D^{*+}

Enddecay

means that the D^{*+} will be not be decayed by EvtGen.

Charge conjugates

CDecay tau+

will define the decays of the tau- to be the charge conjugates of those defined for its charge conjugate (tau+).

- Note: if the tau- decay is later redefined, the tau+ decay modes will not automatically be updated.
- All model arguments will be the same.

“User” decay files

- One or more additional decay files can be read in to override the definitions in DECAY.DEC.
 - Format and available features identical to that of DECAY.DEC
 - Users in BABAR use this feature to define “signal” Monte Carlo requests for their analysis.
- There are a few features of our decay table format that are most commonly used for these decay files.

Particle “aliases”

Alias MyD*+ D*+

Decay B0

1.0 MyD*+ pi- SVS;

Enddecay

Decay MyD*+

1.0 D0 pi+ VSS;

Enddecay

- In this case, all B0s will decay to D*+ pi-, with D*+ → D0 pi+. However, other D*+ in the event will decay as defined in DECAY.DEC.

Charge conjugates for aliased particles

- Often the charge conjugate of an aliased particle will need to be specified.
 - For particles defined in evt.pdl, charge conjugate particle determined from stdhep number
 - For aliases, the charge conjugate must be another alias.

ChargeConj myB+ myB-

- The charge conjugate must be defined for a new models (SSD_CP) and to use the CDecay functionality.

Example of user decay file

Define dgog 0.

Define magqop 1.

Define twobeta -0.92

Define gamma 1.2

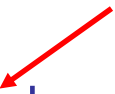
Alias MyB0 B0

Decay Upsilon(4S)

1.0 anti-B0 MyB0 VSS:

Enddecay

dm as defined in
DECAY.DEC



Decay MyB0

1.0 pi+ D*- SSD_CP dm dgog magqop twobeta 1. 0. 0.3 gamma;

Enddecay

End

Control of line shape parameters

- The mass and width can be controlled in the decay table:

Particle rho0 0.779

or

Particle rho0 0.779 0.160

if no width is specified, the default is kept.

Also the minimum and maximum allowed masses are controlled using

ChangeMassMax <particle_name> <min_mass>

ChangeMassMin <particle_name> <max_mass>

Writing new decay models

Writing new Physics Models

- This part of the tutorial deals with writing new models
 - A model is a C++ class that implements the calculation of amplitudes for a given process.
 - This class has to be registered with the frame work in order to be used.
 - The model has a name which is used to indentify the model in the decay table.
- There are currently about 80 decay models implemented in EvtGen.

Some coding history...

- EvtGen was originally started at CLEO
- Most development done at BABAR
 - BABAR has just recently adopted STL
 - EvtGen has started to use STL, but there are still a lot of code that is using bare arrays etc. (BABAR used RW tools.h++, but it was never introduced to EvtGen.)
 - BABAR is soon to adopt standard iostreams. In the public version of EvtGen we have removed the classic iostreams with a script. (We still rely on stringstream which is not part of the C++ standard, but supported in g++.)

States in EvtGen

- EvtGen works with amplitudes. The amplitudes are specified as amplitudes between the initial and final state in a set of basis vector provided by EvtGen.
- EvtGen uses the following representation for the lower spin states:

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	$\rho, J/\Psi$
EvtPhotonParticle	ϵ^μ	1	2	γ
EvtTensorParticle	$T^{\mu\nu}$	2	5	D_2^*, f_2

- Also J=3/2 EvtRaritaSchwinger 4 states
- Higher spin states are represented by a generic helicity state basis

EvtGen support classes for states

- EvtGen provides implementations of several classes that are used to represent the states:
- EvtComplex - implementation of complex numbers
- EvtVector3R, EvtVector3C - real and complex 3-vectors
- EvtVector4R, EvtVector4C - real and complex 4-vectors
- EvtTensor3C, EvtTensor4C - complex second rank tensors
- EvtDiracSpinor - 4-component Dirac spinor
- EvtRaritaSchwinger - Rarita-Schwinger spinor (for spin 3/2 particles)
- EvtGammaMatrix - Dirac gamma matrix implementations

Example decay: $V \rightarrow SS$

To illustrate how a decay model is written we will use the example of the decay of a vector particle to two scalars. The amplitude for this decay is given simply by:

$$A = \epsilon^\mu v_\mu$$

Where ϵ is the polarization vector of the initial vector meson and v is the four-velocity of one of the final state particles.

We will illustrate how we write the class, `EvtVSS`, to implement the calculation of this amplitude for a model named 'VSS'.

The EvtDecayBase class

```
#ifndef EVTDECAYBASE_HH
#define EVTDECAYBASE_HH

#include "EvtGenBase/EvtPatches.hh"
#include "EvtGenBase/EvtId.hh"
#include <string>
#include "EvtGenBase/EvtSpinType.hh"
#include <stdlib.h>
#include <vector>
class EvtParticle;
class EvtSpinType;

class EvtDecayBase{
public:

    //These pure virtual methods has to be implemented
    //by any derived class
    virtual void getName(std::string& name)=0;
    virtual void decay(EvtParticle *p)=0;
    virtual void makeDecay(EvtParticle *p)=0;
    virtual EvtDecayBase* clone()=0;

    //These virtual methods can be implemented by the
    //derived class to implement nontrivial functionality.
    virtual void init();
    virtual void initProbMax();
    virtual std::string commandName();
    virtual void command(std::string cmd);

    ...

};

#endif
```

EvtVSS.hh (simplified)

```
#ifndef EVT_VSS_HH
#define EVT_VSS_HH

#include "EvtGenBase/EvtDecayAmp.hh"

class EvtParticle;

class EvtVSS:public EvtDecayAmp {

public:
    EvtVSS() {}
    virtual ~EvtVSS();

    void getName(std::string& name);
    EvtDecayBase* clone();

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

};
#endif
```

EvtVSS.cc

```
#include <stdlib.h>
#include "EvtGenBase/EvtParticle.hh"
#include "EvtGenBase/EvtGenKine.hh"
#include "EvtGenBase/EvtPDL.hh"
#include "EvtGenBase/EvtVector4C.hh"
#include "EvtGenBase/EvtVector4R.hh"
#include "EvtGenBase/EvtReport.hh"
#include "EvtGenModels/EvtVSS.hh"
#include <string>

EvtVSS::~EvtVSS() {}

void EvtVSS::getName(std::string& model_name){
    model_name="VSS";
}

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

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

void EvtVSS::init(){
    // check that there are 0 arguments
    checkNArg(0);

    // check that there are 2 daughters
    checkNDaug(2);

    // check the parent and daughter spins
    checkSpinParent(EvtSpinType::VECTOR);
    checkSpinDaughter(0,EvtSpinType::SCALAR);
    checkSpinDaughter(1,EvtSpinType::SCALAR);
}

void EvtVSS::decay( EvtParticle *p){

    p->initializePhaseSpace(getNDaug(),getDaug());

    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;
}
```


Registering the model

The last step to do before you can use a model is to register it with the EvtGen framework. This is done in the EvtModelReg.cc:

```
modelist.Register(new EvtVSS);
```

For each instance of a decay in the decay table that uses the VSS model
a new instance of the EvtVSS class is created using the clone method.

Model arguments

Some models takes arguments:

HQET parameters



Decay B0

1.00 D*- e+ nu_e

Enddecay

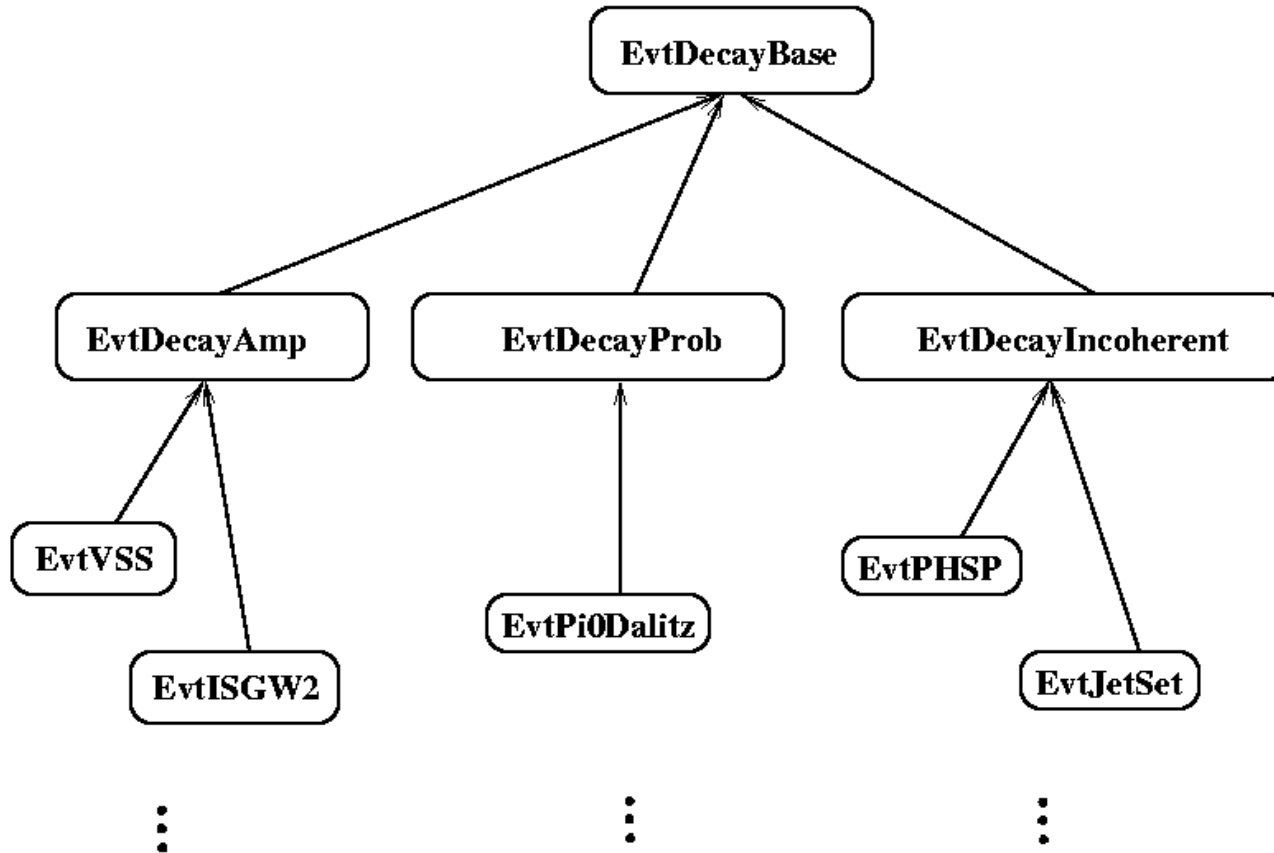
PHOTOS HQET 0.92 1.18 0.72;

These arguments can be accessed in the model using the methods:

`getNArg()` returns the number of arguments

`getArg(i)` returns the *i*th argument

Decay models



Interfacing to EvtGen

Basic EvtGen interface (EvtGen.cc)

```
EvtGen myGenerator(  
    <DECAY.DEC location>,  
    <evt.pdl location>  
    <randomNumberEngine>.  
    <FSR generator>);
```

Optional: PHOTOS
is default.

```
myGenerator.readUDecay(<user decay file>);  
EvtParticle *myParentParticle;  
..... (Set up parent particle properties).....
```

```
myGenerator.generateEvent(myParentParticle,t_init);
```

Setting up random number generator

- Simple interface to use your favorite random number engine:

```
class EvtCLHEPRandomEngine: public  
    EvtRandomEngine {  
public:  
    double random();  
};
```

```
double EvtCLHEPRandomEngine::random() {  
    static HepJamesRandom randengine;  
    return randengine.flat();  
}
```

Initializing parent particle

```
//Find the particle whose name is "B0"  
static EvtId B0=EvtPDL::getId(std::string("B0"));  
  
//The B0 is at rest  
EvtVector4R p_init(EvtPDL::getMass(B0),0.,0.,0.);  
  
//Then create a particle from this 4-vector  
EvtParticle *myParentParticle;  
myParentParticle=  
    EvtParticleFactory::particleFactory(B0,p_init);
```

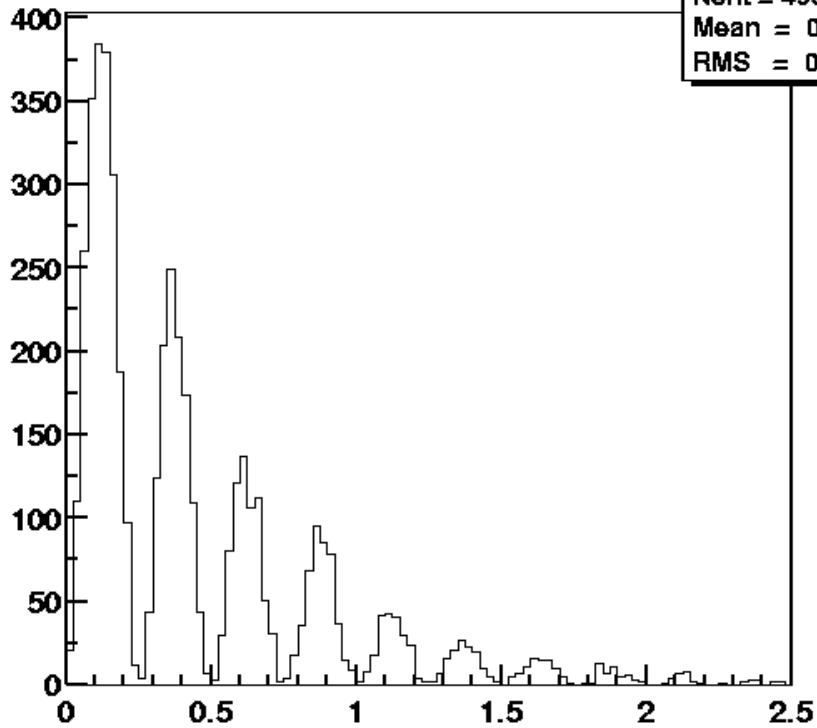
Examples

- EvtGen/testEvtGen.cc is an example of code that interfaces to EvtGen.
 - It is very long only because it has lots of examples.
 - See EvtGen/test/do_tests for examples of how these run. Root scripts exist to make plots from some of these examples.
 - (Documentation very much lacking on this point...ask us for specific info, desired examples)
 - testEvtGen.cc also shows how to override the random # generators in PHOTOS and Pythia.

Topics in Bs from hadronic interactions

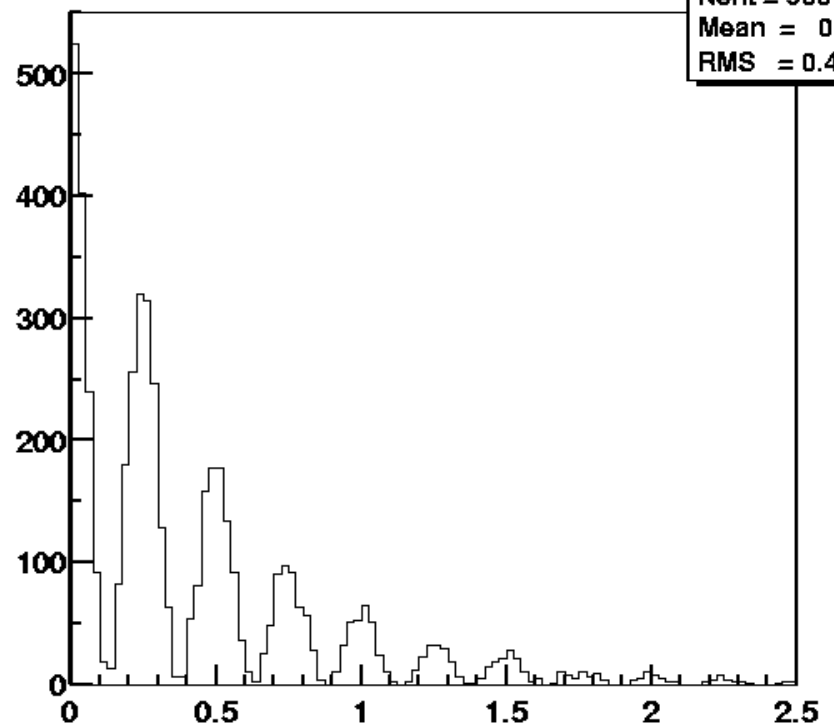
B_s Mixing

Mixed decay (mm)



h1
Nent = 4939
Mean = 0.454
RMS = 0.429

Unmixed decay (mm)



h2
Nent = 5061
Mean = 0.44
RMS = 0.4358

Data vs MC tuning

Data vs MC tuning

- General philosophy on decay table branching fractions:
 - Try to use PDG (or new individual measurements) where possible for exclusive branching fractions.
 - Need to decay all (B) mesons, so significant “interpretation” needed for inclusive BFs.
 - Projections from theory / isospin / other constraints wrt measurements.
 - Educated and/or wild guesses
 - Numbers in the decay table are definitely open to discussion. We hope that we can all agree on the best approach.
 - (MC users will do what they wish anyway...)

Comparisons

- Yesterday showed EvtGen results on $\pi/K/p/\pi^0$ spectra as well as production of various resonances relative to data.
 - Would appreciate feedback on other resonances / measurements that we should pay special attention to in our MC tuning process.

Pythia tuning

- The “inclusive” component of the decay table decayed by Pythia is the primary handle that we have used to improve the data-MC agreement.
- Internal data on eta, eta', omega, and phi resonance production in B decays used during last round of MC tuning. Adjustments of up to 40%.
- No real attempts to tune momentum spectra yet, except for leptons (Fully exclusive decay modes).
 - $B \rightarrow DsX$ study in Belle is an example of what the next step must be.
 - Need to push to get the data for these studies.

Pythia parameters that we have changed

MSTJ(11)	4	Lund fragmentation function
PARJ(41)	0.3	Fragmentation parameters
PARJ(42)	0.58	Same
PARJ(43)	0.50	Same
PARJ(44)	0.90	Same
PARJ(54)	-0.04	Same
PARJ(55)	-0.004	Same
MSTJ(26)	0	Turn off BB mixing in Pythia
PARJ(13)	0.45	D* production tuning (0.6 preferred for ee→qq)
PARJ(26)	0.15	Eta' production
PARJ(25)	1.0	Eta production
PARJ(12)	0.2	Phi production
PARJ(11)	0.4	Phi production
PARJ(33)	0.3	Cut off parameter to stop frag. process
PARJ(14-17)	0.05	Turn on D** production

- Do any of these affect the fragmentation step?
- Should we worry about EvtGen vs. other generator interference?
- Tuning done in very ad hoc fashion. Most handles are indirectly related to the quantities that we measure.