# DecayTreeTuple how to use it.

18-3-2008
Borel Jeremie
Software week

## What it is :

- successor of DecayChainNTuple
- is a DVAlgorithm => must be run within DaVinci, (or know what you do).
- its outcome is a Ntuple with a customizable set of variables ordered in rows
- available since DaVinci v19r9

- DecayTreeTuple itself fills only one variable: “nCandidate”

- Other variables are filled by some “plugins” that the user chooses.

- The “philosophy” is: one line per reconstructed candidate:

| Row number | event number | nCandidate | MyVar-A | MyVar-B |
|---|---|---|---|---|
| 0 | 0 | 0 | 3.2 | -29.3 |
| 1 | **1** | **0** | 3.1 | ... |
| 2 | **1** | **1** | 6.4 | |
| 3 | 2 | 0 | ... | |
| ... | 3 | 0 | | |

- Meant of a given decay with a fixed number of particles
- Not meant for pure Monte Carlo studies.

## A minimalistic example -> the required options!

```
ApplicationMgr.TopAlg +={ "DecayTreeTuple/Dcy" };
Dcy.PhysDesktop.InputLocations = {"Phys/MyRectedGaudiSequencer"};

Dcy.Decay = "[B_s0 -> (^D_s- => ^K+ ^K- ^pi-) pi+]cc";
```

Decay head is stored anyway
(cannot be flagged)

**Flag with a '^' all the particles you want to store in the tuple**

This one will NOT be in the tuple

```
Dcy.TupleName = “MyTuple”;    // optional
```

- You get a root files with a TDirectory named “Dcy”
- It contains a TTree named “MyTuple”
- A lot of variables which were in DecayTreeNTuple are also here by default.

## Adding / removing variables to the tuple:

```
ApplicationMgr.TopAlg +={ "DecayTreeTuple/Dcy" };
Dcy.PhysDesktop.InputLocations = {"Phys/MyRectedGaudiSequencer"};
Dcy.Decay = "[B_s0 -> (^D_s- => ^K+ ^K- ^pi-) ^pi+]cc";

Dcy.ToolList = { “TupleToolKinematic”, “TupleToolTrigger/trig”};
```

Each tool in the ToolList adds some variables to the tuple.

Can be related to the particles, or to the event (e.g. Primary vertices, fill for all candidates).

Tools can be named, their options, can be changed.

```
Dcy.trig.VerboseL0 = true;
Dcy.TupleToolKinematic.OutputLevel = 2;
```

## Naming the particles in the tuple (default behavior):

```
ApplicationMgr.TopAlg +={ "DecayTreeTuple/Dcy" };
Dcy.PhysDesktop.InputLocations = {"Phys/MyRectedGaudiSequencer"};
Dcy.Decay = "[B_s0 -> (^D_s- => ^K+ ^K- ^pi-) ^pi+]cc";
```

All particle related variables in the tuple are named with a prefix.
By default, a “sanitized” version of their name.

Examples:
B_s0_ID
piminus_TRUEP_Z, piplus_TRUEP_Z
D_sminus_PX

Use it with care (merging)!!
The **first** event you reconstruct sets the column names,
if it is B_s0~ -> (D_s+ => K- K+ pi+) pi- the two names get swapped
(actually this is a bug that should be corrected...)

Quick fix through options:

```
Dcy.UseLabXSyntax = true;
```

lab0_TRUEP_Z, lab1_TRUEP_Z, etc...
consistency is (in principle) guaranteed, but unfriendly syntax.

## Naming the particles in the tuple (the better way):

use the Branches map:

```
ApplicationMgr.TopAlg +={ "DecayTreeTuple/Dcy" };
Dcy.PhysDesktop.InputLocations = {"Phys/MyRectedGaudiSequencer"};
Dcy.Decay = "[B_s0 -> (^D_s- => ^K+ ^K- ^pi-) ^pi+]cc";

Dcy.Branches = {
    "kaon"        : "[B_s0 -> (D_s- => ^K+ ^K- pi- ) pi+]cc"
   ,"Ds"          : "[B_s0 -> (^D_s- => K+ K- pi- ) pi+]cc"
   ,"bachelor"    : "[B_s0 -> (D_s- => K+ K- pi-) ^pi+]cc"
   ,"Bs"    : "[B_s0]cc :[B_s0 -> (D_s- => K+ K- pi- ) pi+]cc"
    };
```

Flagging the head is slightly exotic.

Don't forget this kind of features to flag all the possible decays.

Will name the particles after
*kaon0_ID, kaon1_ID, bachelor_PX*, etc...
Unflagged particles named after their default values.

The **`Decay`** property is still required to tell which particles are filled.

## Fine tuning of the filled variables:

Add variables to be filled only for some particles in the decay:

```
...
Dcy.Branches = {
   "kaon"   : "[B_s0 -> (D_s- => ^K+ ^K- pi- ) pi+]cc"
  ,"Ds"     : "[B_s0 -> (^D_s- => K+ K- pi- ) pi+]cc"
  ,”Bs”     : “[B_s0]cc :[B_s0 -> (D_s- => K+ K- pi- ) pi+]cc"
   };

Dcy.Ds.ToolList = { "TupleToolMCBackgroundInfo/dsBkg" };
Dcy.Ds.dsBkg.OuputLevel = 3;
Dcy.Ds.InheritTools = false;
```

No tools from `Dcy.ToolList` will be executed on the Ds.

One can fills BackgroundCat info only for the Ds.

Or intantiate twice, for the Bs and the Ds and give different options.

## The easy way to know the actual branch names:

Look at your log, at the very first event filled, DecayTreeTuple prints:

```
Dcy       INFO Tree Dcy initialized:
Event related tools: TupleToolEventInfo
Particle related stuffers: realname (tuplename)
|B_s~0 (Bs)          :TupleToolPid, TupleToolMCPid, TupleToolPropertime
|   D_s+ (Ds)        :TupleToolPropertime
|      K+ (Ds_dau1)  :TupleToolMCPid, TupleToolPid
|      K- (Ds_dau2)  :TupleToolMCPid, TupleToolPid
|      pi+ (Ds_dau0) :TupleToolMCPid, TupleToolPid
|   pi- (H)          :TupleToolMCPid, TupleToolPid
```

Tuple column prefix

Rect'ed particle name

Tools associated to this particles

## Finding doc on the tools, which tools now exist, finding a variable's tool:

DecayTreeTuple is a package in itself. The “official” tools are also in this package

```
cd $DECAYTREETUPLEROOT/src/
ls *.h
TupleToolGeometry.h          TupleToolMCBackgroundInfo.h
TupleToolPid.h               TupleToolTagging.h
DecayTreeTuple.h             TupleToolMCPid.h
TupleToolPrimaries.h         TupleToolTrigger.h
TupleToolKinematic.h         TupleToolMCTruth.h
TupleToolPropertime.h        TupleToolEventInfo.h
```

Then look at their doxygen doc or read their code.
(grep or Lbglimpse to find a variable suffix).

At the time, all the tools are named `TupleToolAName`, but be sure this kind of convention will not hold over months (weeks?)... hence look what is in the package.

This package is meant (quite like all the others but..) to be maintained by the users
=> it will quickly become a bit chaotic.
=> it is worth (and is the least for a real analysis!) to spend some time looking at the tool code and understand which variables you use.

## Writing your own variables in the tuple:

a) getpack Phys/DecayTreeTuple vXrY (v1r1 at the time)

b) create GaudiTool deriving from

1) IParticleTupleTool.h -> for particle related variables

or 2) IEventTupleTool.h -> for event related variables

=> probably even easier, copy some of the existing one.

c) define the method

Decay head

Particle to fill info about

Variable name prefix

```
    1) StatusCode fill(  const LHCb::Particle*
                         , const LHCb::Particle*
                         , const std::string&
                         , Tuples::Tuple& );

    or 2) StatusCode fill( Tuples::Tuple& );
```

## Writing your own variables in the tuple (an example in two lines):

- Look at all the examples in `$DECAYTREETUPLEROOT`

Your code:

```
StatusCode myTool::fill( const Particle*
                       , const Particle* part
                       , std::string& prefix
                       , Tuples::Tuple& tpl ){
   StatusCode sc = tpl->colum( prefix+"_MyMomentum", part->p() );
   return sc;
}
```

But unphysical default value is ok, and even recommended.

Return StatusCode::FAILURE if one variable cannot be filled.
This prevents the present candidate to be written in the tuple.

Your options:

```
Dcy.ToolList += { "myTool" };
```

That's it, you got all your particle momenta.

=> Never change the number of column in the tuple

a) Make sure that you either always or never fills the variables

b) Don't call `tpl->write();` yourself

DecayTreeTuple is now released. It is meant to provide a quick, flexible and not too dirty way to have a Tuple filled.

- Look at the doxygen doc. It is more likely to evolve than these slides...

- Not a monolithic dinosaur anymore.

=> Will remain maintainable but will probably become slightly chaotic as the number of available tool grows.

Now, instead of mailing about a problem in your “private version of DecayChainNTuple”, you can mail about a problem in “DecayTreeTuple with TupleToolTrigger v2r4, TupleToolBackground v5r8, private version of TupleToolKinematic, etc...”
Much simpler obviously :)

- As the code segment actually filling the variables are well localized, and hopefully small, should be easier to understand, debug and adapt.

- A first set of variables are available. Hope it will grow with the time (don't wait, code them... missing tools are: event collection, tagging, surely many others...)