LHCb Note 2005-016, OFFLINE
LPHE Note 2005-012
January 17, 2006

DaVinci for Busy People

Generic selection algorithms — a user guide

Version 2 (DaVinci v12r15)

P. Koppenburg[1]
CERN

L. Fernández[2]
LPHE, EPFL

Abstract

This note describes generic selection algorithms and tools allowing to perform a complete physics selection in DaVinci only using options. Although this way is not optimal for complicated analyses, it allows to get results quickly.

The tools described here have been used successfully in the exclusive part of the high-level-trigger [1] and start to be used in pre-selections for the stripping.

Make sure you read the warning in Section 5 before you start typing any option file.

This version of the note refers to DaVinci v12r15 and v14r5.

[1] Patrick.Koppenburg@cern.ch
[2] Luis.Fernandez@epfl.ch

Contents

1 Introduction

Many typical analyses in DAVINCI can be steered only by options using generic algorithms. For instance the exclusive high-level-trigger HLT [1] and some pre-selections in the Data Challenge DC'04 Stripping are based on generic selection algorithms configured by options. This is an easy way of getting quickly results out of DAVINCI. It avoids duplication of code and ensures that only well tested algorithms and tools are used. It also encourages users who find a missing feature to write code to be released with DAVINCI rather than developing their private over-specialised algorithm.

However we would like to make the distinction between *selection* and *analysis*. This generic approach is meant to be for (pre-)selection purposes and may not be suited for a fine-tuned analysis where the user will have to code anyway his own final algorithm, especially for monitoring or fitting purposes. The present approach is for instance particularly well suited for analyses using maximum-likelihood fits (like ${}_s\mathcal{P}lot$ [2]) where one would not apply any hard cut in DAVINCI but extracts the signal from a fit to many variables.

There are several ways to quickly get a physics result:

Plain `C++`: `DVAlgorithm` inherits from `GaudiAlgorithm` (and `GaudiTupleAlg` ...): a lot of typing is saved.

LOKI: "Loops and Kinematics" [3] is a metalanguage based on templated `C++`, with even more typing saved.

BENDER: Interactive `python` using LOKI.

Generic algorithms: The subject of this note.

The common assumption is that physicists always do the same, hence any line of `C++` you type is a duplication of what your office-mate has been typing yesterday.

Most of (B-) physics analyses are a sequence of A $\to$ B C (...), with some cuts in between.

The minimal information a selection algorithm needs is:

1. Where to get the particles;
2. What decay to reconstruct;
3. What cuts to apply;
4. Where to put the data.

While items 1 and 4 are handled by options in all use-cases,[3] the decays and the cuts are usually defined in the code. This is where generic algorithms propose a different approach, the decays and cuts being also defined in options (see Section 5 for a warning about "programming by options").

For instance to reconstruct a D_s, one could write some `C++` code in order to loop over three vectors of particles, the K^+, K^- and $\pi^\pm$:

[3] The input location is a property of the `PhysDesktop`. The output is enforced to be `/Event/Phys/<AlgoInstanceName>/`.

```
for( ParticleVector::const_iterator mK = KMinus.begin() ;
     mK != KMinus.end() ; ++mK ){
  for( ParticleVector::const_iterator pK = KPlus.begin() ;
       pK != KPlus.end() ; ++pK ){
    for( ParticleVector::const_iterator pi = Pions.begin() ;
         pi != Pions.end() ; ++pi ) {
      [...] } } }
```

In LoKi, this is much shorter:

```
for( Loop Ds = loop("K K pi", "D_s+", FitVertex); Ds; ++Ds ){ [...] }
```

However, once the (anyway mandatory) decay descriptor is given, all the information is there:

```
DsForBs2DsPi.DecayDescriptor = "[D_s+ -> K+ K- pi+]cc";
```

This is the approach described in the present note.

1.1 History

The first generic selection algorithms have been written by Gerhard Raven. These are:

Select2ParticleDecay: Makes decays to two (and more) particles. Used in DC'03 for $\mathrm{B_s} \to \mathrm{J}/\psi\phi$.

RefineSelection: Allows to "refine" a set of particles applying cuts.

CombineParticles: Replaces `Select2ParticleDecay` with a better syntax of options.

We decided to use them in the exclusive HLT. Then it appeared that:

- The option syntax is incompatible between `RefineSelection` and `CombineParticles`; consequently a quick ("cut-and-paste") reshuffling of options is not straightforward;
- `CombineParticles` is too slow, essentially because the vertex fitting is done before the mass cut is applied.

These algorithms are now considered as obsolete and will disappear from DaVinci in the next backward-incompatible versions.

1.2 New implementation

These first generic algorithms have been replaced recently by the following ones:

The MakeResonances algorithm: Yet another `CombineParticles`.

The FilterDesktop algorithm: A `RefineSelection` with a similar syntax than `MakeResonances`.

The ByPIDFilterCriterion tool is used by `MakeResonances` and `FilterDesktop` to apply all cuts, ensuring a coherent syntax.

The IPlotTool: The algorithms described above use tools interfacing `IPlotTool` for a quick plotting of some variables. There are two implementations:

The SimplePlotTool: Makes plots of any given set of variables for any particle;

The RecursivePlotTool: Calls `SimplePlotTool` for each particle and its daughters recursively.

This new code is used since December 2004 in the HLT, together with a series of filter criteria. It is also used in the `Bd2DPi`, `Bs2PhiEtac`, `Bs2JpsiEta`, `Bu2LLK` and `Bu2JpsiK` selections. selection. It is described in Section 3.

Additionally a new algorithm `DecayChainNTuple` allows to fill a quite complete N-tuple for the selected decay chain (and the Monte-Carlo MC truth). It is described in Section 4.

2 A Complete Example: $B^0 \to D^- \pi^+$

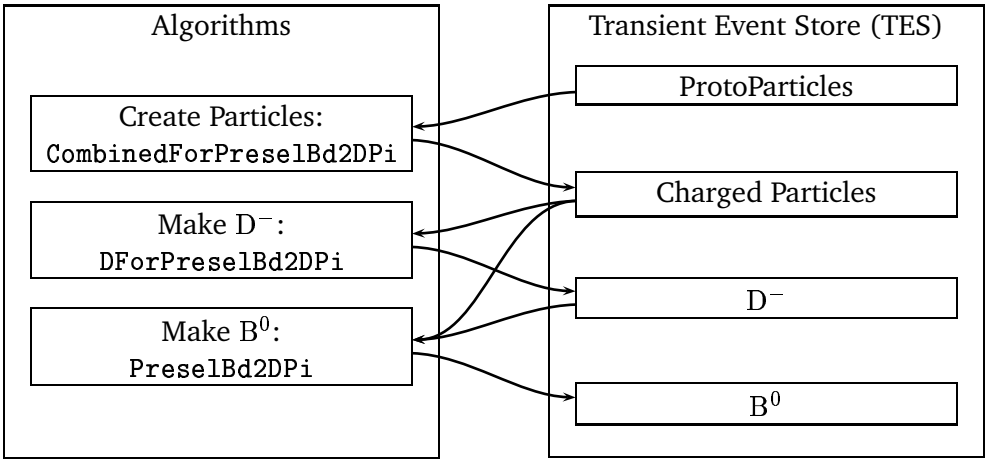

Figure 1: Design of the Algorithmic Sequence

The usage of these algorithms and tools is illustrated by Vladimir Gligorov's preselection package `Bd2DPi` for the decay $B^0 \to D^- \pi^+$ and its corresponding charge conjugate (*cc*). We suggest to clearly separate the distinct operations, rather than writing a single monolithic algorithm for the whole decay chain. This is illustrated in Figure 1. The three steps are:

1. Create `Particles` from the `ProtoParticles` (i.e. assign a PID);
2. Build the $D^- \to \pi^-\pi^-K^+$ decay and *cc*;
3. Build the $B^0 \to D^- \pi^+$ decay and *cc*.

For each of these steps we use a different algorithm *instance*, embedded in a `GaudiSequencer`:

```
ApplicationMgr.TopAlg+={"GaudiSequencer/SeqPreselBd2DPi"};
SeqPreselBd2DPi.Members += {"PreLoadParticles/CombinedForPreselBd2DPi",
                            "MakeResonances/DForPreselBd2DPi",
                            "MakeResonances/PreselBd2DPi",
                            "PrintHeader/PrintPreselBd2DPi"};
```

There are actually two algorithms involved here:

- One `PreLoadParticles` that calls the `CombinedParticleMaker`;
- Two instances of `MakeResonances`: one to form the D^- and one to make the B^0.

The somewhat longish instance names `CombinedForPreselBd2DPi`, `DForPreselBd2DPi`, `PreselBd2DPi` and `PrintPreselBd2DPi`[4] are required by the DC'04 Stripping guidelines [4] to avoid confusion.[5]

2.1 Particle making

```
CombinedForPreselBd2DPi.PhysDesktop.ParticleMakerType =
                          "CombinedParticleMaker";
CombinedForPreselBd2DPi.PhysDesktop.CombinedParticleMaker.Particles =
                         {"kaon", "pion" };
// [...] some more cuts
```

`MakeResonances` is a `DVAlgorithm` and hence can make particles, but it is recommended to make particles in a separate algorithm and to re-use them when needed.

These particles are given as input of `DForPreselBd2DPi`:

```
DForPreselBd2DPi.PhysDesktop.InputLocations =
                                    { "Phys/CombinedForPreselBd2DPi" };
```

2.2 The decay $\mathrm{D}^- \to \pi^-\pi^-\mathrm{K}^+$

```
DForPreselBd2DPi.DecayDescriptor = "[D- -> pi- pi- K+]cc" ;
DForPreselBd2DPi.Window = 50.*MeV ;
```

These two lines of options (together with the declaration of the `DForPreselBd2DPi` algorithm in the sequencer above) are enough to make all $\pi^-\pi^-\mathrm{K}^+$ combinations in a mass window of ± 50 MeV around the nominal mass of the D^-:

- The `DecayDescriptor` option tells the algorithm what decay to reconstruct;
- The mass window is applied on the sum of 4-vectors before vertex fitting;
- The vertex fitter applied is the `UnconstrVertexFitter`. So far this cannot be changed.[6]

2.3 Some cuts

`MakeResonances` owns two private instances of the `ByPIDFilterCriterion` tool, one for the daughters (`DaughterFilter`), and one for the mother (`MotherFilter`). Here are the cuts for the daughters:

[4] The `PrintHeader` algorithm only prints a nice message for selected events when `PrintPreselBd2DPi.OutputLevel = 2`.

[5] That would inevitably happen in a stripping environment if we had called the algorithms `MakeD` and `MakeB`.

[6] Until we revise the vertex fitters interface.

```
DForPreselBd2DPi.DaughterFilter.Selections = {
                "K+  : KinFilterCriterion, PVIPFilterCriterion",
                "pi+ : KinFilterCriterion, PVIPFilterCriterion"};
DForPreselBd2DPi.DaughterFilter.KinFilterCriterion.MinPt = 300.*MeV ;
DForPreselBd2DPi.DaughterFilter.PVIPFilterCriterion.MinIPsignif = 1. ;
```

These options apply a 300 MeV transverse momentum p_T and a 1σ impact parameter IP cuts to the kaons and pions before making the D^-. If one wants to apply different cuts for the K and the π, one simply needs to give different instance names to the appropriate filter tools, as in:

```
DForPreselBd2DPi.DaughterFilter.Selections =
                             { "K+  : KinFilterCriterion/KKin,
                                      PVIPFilterCriterion/KPVIP",
                               "pi+ : KinFilterCriterion/PiKin,
                                      PVIPFilterCriterion/PiPVIP"};
DForPreselBd2DPi.DaughterFilter.KKin.MinPt = 300.*MeV ;
DForPreselBd2DPi.DaughterFilter.KPVIP.MinIPsignif = 1. ;
DForPreselBd2DPi.DaughterFilter.PiKin.MinPt = 500.*MeV ;
DForPreselBd2DPi.DaughterFilter.PiPVIP.MinIPsignif = 2. ;
```

2.4 Cuts on the $D^{\pm}$

Then one uses the `MotherFilter` to apply cuts on the $D^{\pm}$:

```
DForPreselBd2DPi.MinPt = 2000.*MeV ;
DForPreselBd2DPi.MotherFilter.Selections =
           { "D+ : VtxFilterCriterion, PVIPFilterCriterion,
                   FlightDistanceFilterCriterion/FDCut" };
DForPreselBd2DPi.MotherFilter.VtxFilterCriterion.MaxChi2 = 20. ;
DForPreselBd2DPi.MotherFilter.PVIPFilterCriterion.MinIPSignif = 2. ;
DForPreselBd2DPi.MotherFilter.FDCut.CutBestPV = true;
DForPreselBd2DPi.MotherFilter.FDCut.MinFSPV = 4.5 ;
```

These options select $D^{\pm}$ with a $p_T > 2$ GeV, a vertex $\chi^2 < 20$, a 2σ IP on any primary vertex PV and a 4.5σ flight separation from the PV to which it "points the best".[7]

Like the mass `Window`, the `MinPt` cut is a property of `MakeResonances` and is applied before the vertex fit. It's quite helpful for the HLT. It does the same as:

```
DForPreselBd2DPi.MotherFilter.Selections = { "D+ : KinFilterCriterion,
DForPreselBd2DPi.MotherFilter.KinFilterCriterion.MinPt = 2000.*MeV ;
```

but much faster!

[7] The PV is defined as the primary to which the particle being filtered has the smallest IP significance.

2.5 Make the B^0

Selecting the B^0 is the same game. Here we show all options in one box:

```
PreselBd2DPi.PhysDesktop.InputLocations = {"Phys/DForPreselBd2DPi",
                                    "Phys/CombinedForPreselBd2DPi"};
PreselBd2DPi.DecayDescriptor = "[B0 -> D- pi+]cc" ;
PreselBd2DPi.DaughterFilter.Selections = { "pi+ : KinFilterCriterion,
                                             PVIPFilterCriterion" };
PreselBd2DPi.DaughterFilter.KinFilterCriterion.MinPt = 500.*MeV ;
PreselBd2DPi.DaughterFilter.PVIPFilterCriterion.MinIPsignif = 2.5 ;
PreselBd2DPi.Window = 500.;
PreselBd2DPi.MotherFilter.Selections = { "B0 : VtxFilterCriterion,
                                              PVIPFilterCriterion,
                    Momentum2FlightAngleFilterCriterion/Mom2Flight" };
PreselBd2DPi.MotherFilter.VtxFilterCriterion.MaxChi2 = 20. ;
PreselBd2DPi.MotherFilter.PVIPFilterCriterion.MaxIPsignif = 6.;
PreselBd2DPi.MotherFilter.Mom2Flight.CosAngle = 0.999;
```

We require the bachelor π to have a p_T in excess of $500\,\mathrm{MeV}$, an IP of 2σ and build a B^0 in a mass window of $\pm 500\,\mathrm{MeV}$. The B^0 candidate has to have a vertex $\chi^2 < 20$, an IP larger than 6 times its error and the angle α of its flight direction to the momentum must satisfy $0.999 < \cos\alpha$.

This ends the whole pre-selection: only 38 lines of options!

2.6 A few plots

Finally one can switch on histogramming in any algorithm with, for instance for the final B^0 and all its descendants:

```
PreselBd2DPi.HistoProduce = true ;
PreselBd2DPi.DaughterPlots.Variables =
                           { "M", "Chi2", "Vz", "P", "Pt", "IPs"} ;
PreselBd2DPi.MotherPlots.Variables = { "WM", "M", "Chi2", "Vz", "P",
                                       "Pt", "IPs", "FS" } ;
PreselBd2DPi.DaughterPlotsPath = "BdIn" ;
PreselBd2DPi.MotherPlotsPath   = "BdOut";
```

3 Generic Selection Algorithms Reference

3.1 `MakeResonances` reference

MakeResonances

`DVAlgorithm` Author: *Patrick Koppenburg*
In `Phys/DaVinciTools`

Algorithm that reconstructs a decay according to a given decay descriptor.

Options:

`std::string DecayDescriptor`	Defines the decay to reconstruct. Actually a property inherited from `DVAlgorithm`.
`std::vector<std::string> DecayDescriptors`	Array of decay descriptors, overrules `DecayDescriptor`.
`double Window`	Mass half-window applied on the sum of 4-vectors, *before* vertexing.
`double LowerWindow`	Lower mass half-windows for asymmetric mass cuts.
`double UpperWindow`	Upper mass half-windows for asymmetric mass cuts.
`double MinMomentum`	Momentum cut applied on the sum of 4-vectors, *before* vertexing
`double MinPt`	p_T cut applied on the sum of 4-vectors, *before* vertexing
`bool MotherToNGammas = false`	Creates a composite particle at the origin only using photons.
`bool KillOverlap = true`	Discard combinations with the same `ProtoParticle` used more than once (uses the `CheckOverlap` tool).
`std::string DaughterFilterName = "DaughterFilter"`	Name of the filter applied to daughters.
`std::string MotherFilterName = "MotherFilter"`	Name of the filter applied to mothers.
`bool HistoProduce = false`	Make plots. Actually a property inherited from `GaudiHistoAlg`.
`std::string DaughterPlotTool = "RecursivePlotTool/DaughterPlots"`	Name of the plot tool applied to the daughters.
`std::string MotherPlotTool = "RecursivePlotTool/MotherPlots"`	Name of the plot tool applied to the mother.
`std::string DaughterPlotsPath = ""`	Path for daughter plots.
`std::string MotherPlotsPath = ""`	Path for mother plots.

The syntax understood by the `DecayDescriptor` is very simple:

```
HLTSharedPhi.DecayDescriptor = "phi(1020) -> K+ K-" ;
HLTSharedKstar.DecayDescriptor = "[K*(892)0 -> K+ pi-]cc" ;
```

where `[<Decay>]cc` means that both the given decay and its charge conjugate will be

reconstructed. The DecayDescriptor property does not follow the much more elaborated decay descriptor syntax of the (MC)DecayFinder tool. For instance don't try

```
[B_s0 -> (J/psi(1S) -> mu+ mu- {,gamma} {,gamma})
         (eta -> pi+ pi- ( pi0 -> gamma gamma)]cc
```

The option DecayDescriptors allows to use the same algorithm to reconstruct several similar decays. For instance:

```
// Actual reconstructed decays:
DiLeptonForPreselBu2LLK.DecayDescriptors = {"J/psi(1S) -> mu+ mu-" ,
                                            "J/psi(1S) -> e+ e-"};
// Only used in SelResult container:
DiLeptonForPreselBu2LLK.DecayDescriptor = "J/psi(1S) -> l+ l-" ;
```

If you are planning to use MakeResonances in decays involving photons, please read Appendix A.

3.2 FilterDesktop reference

FilterDesktop allows to select particles from a given location in the TES and clones the particles before saving them to another location in the TES.

Practically this algorithm is seldom used since all selection cuts can be applied in MakeResonances. It has yet some use-cases in the HLT. It can for instance be useful for selections with large combinatorics, for instance to apply a pre-selection on particles before starting to combine them.

Warning: The cloning of particles implies that a *different* object is created and hence the MC association of original particles cannot be re-used for the cloned particles. The association has to be re-run on the cloned particles.

FilterDesktop

DVAlgorithm — Author: *Patrick Koppenburg*
In Phys/DaVinciFilter

Algorithm that selects particles from a given location according to some cuts.

Options:

bool HistoProduce = false	Make plots. Actually a property inherited from GaudiHistoAlg.
std::string InputPlotTool = "RecursivePlotTool/InputPlots"	Name of the plot tool applied to daughters.
std::string OutputPlotTool = "RecursivePlotTool/OutputPlots"	Name of the plot tool applied to mothers.
std::string InputPlotsPath = ""	Path for daughter plots.
std::string OutputPlotsPath = ""	Path for mother plots.

3.3 ByPIDFilterCriterion reference

The ByPIDFilterCriterion handles all cuts in MakeResonances and FilterDesktop. It applies a list of FilterCriterion on the input particles, depending on their PID.

ByPIDFilterCriterion

GaudiTool, Interface: IFilterCriterion Author: *P. Koppenburg*
In Phys/DaVinciFilter

Returns a yes/no depending on a list of criteria for each PID.
Allows to filter composite particles according to criteria applied to its descendants.

Options:

`std::vector<std::string> Selections`	List of Selections.
`bool ApplyCC = true`	Use same selection for particle and anti-particle.
`bool ExclusiveSelection = false`	Filter out particles not explicitly given in `Selections`.
`bool FilterByDescendents = false`	Use descendants to filter composite particles.

Note that `ExclusiveSelection` and `FilterByDescendents` cannot be true both at the same time.

The syntax of `Selections` is

```
...Selections = { " P1 : AAAFilterCriterion/Instance1,
                         BBBFilterCriterion/Instance2, [...] ",
                  " P2 : CCCFilterCriterion/Instance3,
                         DDDFilterCriterion/Instance4, [...] ",
                   [...] };
```

Where `P1`, `P2` are particle names (`mu+`, `J/psi(1S)` ...),[8] and `XXXFilterCriterion` are `IFilterCriterion` tools. The instance names are optional. The same tool instance can be applied to particles with different PID if one wants the same cut to be applied. There is no limitation in the number of `IFilterCriterion` or particle names.

The `" P1 : AAAFilterCriterion/Instance1, [...] "` strings currently support spaces, line breaking and other special characters like tabulations (they are all ignored). They don't support comments:

```
...Selections = { " mu+ : KinFilterCriterion, // apply PT cut
                          PVIPFilterCriterion" };
```

would result in an error. Generally all syntax errors cause an interruption at initialisation.

3.4 Filter criteria

There are several filter criteria available. Please refer to the `Doxygen` documentation for the various options. The latest list of available filters can be obtained by looking at the `IFilterCriterion` documentation.

ConstrainedChi2FilterCriterion: Performs a mass-constrained vertex fit and cuts on the χ^2 of this new vertex. The vertex is not saved (*Besma M'charek*).

DLLFilterCriterion: Cuts on the delta-log-likelihood of the particle ID (*Jan Amoraal*).

[8] The list of `ascii` names of the different particles used by the `ParticlePropertySvc` can be found in the package `ParamFiles`, file `ParticleTable.txt`.

FlightDistanceFilterCriterion: Cuts on the flight distance of a composite from the "best" primary vertex (*Luis Fernández*).

KinFilterCriterion: Momentum and p_T cuts (*Paul Colrain*).

LifetimeSignificanceFilterCriterion: Cuts on the flight distance to PV in units of proper time (*Gerhard Raven*).

MassDifferenceFilterCriterion: Mass difference cuts (*Gerhard Raven*).

MassFilterCriterion: Mass cuts (*Gerhard Raven*).

Momentum2FlightAngleFilterCriterion: Cuts on the alignment of momentum with direction of flight, assuming the particle comes from the PV (*Luis Fernández*).

MomentumMotherDirectionFilterCriterion: Cuts on the angle of the particle's direction with respect to the momentum of its mother (*Federica Legger*).

OverlapFilterCriterion: Applies `CheckOverlap` tool (*Patrick Koppenburg*).

PIDFilterCriterion: Selects particles of a given PID of a particle. It also allows to cut on the confidence level of the assigned particle ID. For more sophisticated PID cuts use the `DLLFilterCriterion` (*Paul Colrain*).

PVIPFilterCriterion: Impact parameter cuts (*Patrick Koppenburg*).

TrackTypeFilterCriterion: Track type requirements (*Patrick Koppenburg*).

VtxFilterCriterion: Vertex χ^2 and position cuts (*Gerhard Raven*).

VtxIsolationFilterCriterion: Checks that the decay vertex is isolated from any number of tracks, according to χ^2 difference or IP requirements (*Luis Fernández*).

BooleanFilterCriterion: Allows any logical combination of `IFilterCriterion` tools (*Gerhard Raven*).

ByPIDFilterCriterion: A PID-dependent logical combination of `IFilterCriterion` tools. Don't forget about the possibility to re-use it inside a `ByPIDFilterCriterion` (*Patrick Koppenburg*).

TrueMCFilterCriterion: Keeps particles associated to a given true MC decay (*Patrick Koppenburg*).

Generally filter criteria using primary vertices will apply separation cuts with respect to any primary vertex, or pick up the PV to which the particle point most likely for all other cuts.

3.5 Plotting tools

The generic plotting tools are in a very preliminary stage, but useful enough for the HLT. They are based on the interface `IPlotTool`.

3.5.1 The SimplePlotTool

SimplePlotTool

GaudiHistoTool, Interface: IPlotTool — Author: *Patrick Koppenburg*
In Phys/DaVinciTools

Produces plots according to the given variables.

Options:

std::vector<std::string> Variables	Variables to plot.
std::vector<double> Minima	Lower limits of histograms
std::vector<double> Maxima	Upper limits of histograms

For the Variables, the following are defined: M (Mass), WM (Wide Mass), DM (Mass Difference), P, Pt, Chi2, IP, IPs (IP significance), DPV (Distance to PV), FS (Flight distance Significance), Vz , Vr , Vx , Vy (Vertex coordinates).[9]

A different plot is produced for each particle of different PID, in the order the particles appear. There is thus no way to have the histograms "ordered".

The boundaries are set automatically (depending on the PID). One can set them using the Minima and Maxima options, but unfortunately it is mandatory to give ranges to either none or all variables.

3.5.2 The RecursivePlotTool

The RecursivePlotTool loops over all descendants of the particles it has to plot and calls the SimplePlotTool recursively. It has the same options as the SimplePlotTool (it actually overwrites the options of the latter).

3.5.3 The SimplePlots algorithm

There is an algorithm SimplePlots that does nothing else than calling an IPlotTool:

```
SimplePlots.PhysDesktop.InputLocations = {"Phys/SomeLocation"};
SimplePlots.PlotTool = "RecursivePlotTool/Plots" ;
SimplePlots.Plots.Variables = { "M", "Chi2" } ;
```

By default the PlotTool option is set to SimplePlotTool/Plots.

4 Generic N-tuple Algorithm: DecayChainNTuple

In this section we describe the DecayChainNTuple algorithm, which saves all kind of information (with optional MC truth) to a N-tuple for a given selected decay chain.[10] This generic algorithm can be configured by options, allowing to get quickly most of the interesting variables stored to a N-tuple.

[9] It is planned to change these variable names to conform with the LoKi shortcuts for consistency.
[10] Is is recommended to use ROOT.

We illustrate the use of `DecayChainNTuple` with a simple example, leaving the algorithm reference for later. We consider in our example the selection of the $\mathrm{B_s} \to \mathrm{J}/\psi(\mu^+\mu^-)\phi$ $(\mathrm{K^-K^+})$ decay channel done in the following sequence:

```
ApplicationMgr.TopAlg += {"GaudiSequencer/SeqProcessBs2JpsiPhi"};
SeqProcessBs2JpsiPhi.Members += {"PreLoadParticles/CombinedForBs2JpsiPhi",
                                 "MakeResonances/Jpsi2MuMuForBs2JpsiPhi",
                                 "MakeResonances/Phi2KKForBs2JpsiPhi",
                                 "MakeResonances/Bs2JpsiPhi",
                                 "DecayChainNTuple/OffBs2JpsiPhiTree"};
```

where `OffBs2JpsiPhiTree` is an instance of `DecayChainNTuple`.

The default configuration is that of an *offline* selection such that the user just needs a few lines to get the N-tuple. What `DecayChainNTuple` needs to know is:

- Where to look for the particles:

```
OffBs2JpsiPhiTree.PhysDesktop.InputLocations = {"Phys/Bs2JpsiPhi"};
```

- What decay to reconstruct:

```
OffBs2JpsiPhiTree.Decay =
"B_s0 -> (^J/psi(1S) -> ^mu+ ^mu-) (^phi(1020) -> ^K+ ^K-)";
```

This will look for all the selected $\mathrm{B_s}$ candidates according to the required decay chain, which is set using the `Decay` property, and save a N-tuple with the default name being `FILE1/MySelection`. The user can specify a different N-tuple name with

```
OffBs2JpsiPhiTree.NtupleName = "FILE1/OffBs2JpsiPhi";
```

The syntax used for the decay chain is that of the `DecayFinder` with a hat flag (`^`) in front of the particles for which one wishes to have information. Note that the mother of the decay (here the $\mathrm{B_s}$) is always booked.

4.1 Variables

For any standard GAUDI algorithm, and hence for a `DVAlgorithm`, one needs to declare all the N-tuple's items in the header file and then book the N-tuple with the previously declared items. In order to overcome this feature and as in `DecayChainNTuple` the number of particles depends on the decay under study, some labelling of the N-tuple's variables is made:

- `_lab0` is appended to the names of the variables related to the mother of the decay;
- `_lab1, _lab2, ...` is appended to the names of the variables related to the (flagged) daughters.

In this way all the items are declared only once and the N-tuple is booked when the first decay of interest is found.

The labelling relies on the `DecayFinder` grammar and syntax which read the decay string from right to left, starting at the sub-head. In our example we will have the following labels:

```
        B_s0 -> (^J/psi(1S) -> ^mu+ ^mu-) (^phi(1020) -> ^K+ ^K-)
labels: 0         4             6    5      1             3   2
```

For simplicity, the labels are also printed once in the log file when the N-tuple is booked:

```
Booking ParticleName (mother) B_s0
Booking Subdaughter number = 1 , ParticleName phi(1020)
Booking Subdaughter number = 2 , ParticleName K-
Booking Subdaughter number = 3 , ParticleName K+
Booking Subdaughter number = 4 , ParticleName J/psi(1S)
Booking Subdaughter number = 5 , ParticleName mu-
Booking Subdaughter number = 6 , ParticleName mu+
```

There is a large number of N-tuple variables automatically saved: have a look at the source file to know what variables are defined and their type.[11] A few examples are:

- `mass_lab0`: mass of the reconstructed $\mathrm{B_s}$;
- `vchitwo_lab1`: χ^2 of ϕ vertex;
- `pt_lab6`: transverse momentum of one of J/ψ daughters;
- `nRecoPV`: number of reconstructed primary vertices.

Note that all the variables related to the particles (i.e. with a `_lab`) are arrays indexed by the number of selected mother candidates. `DecayChainNTuple` also has the possibility to look for different decays at the same time:

```
OffTreeB2HH.Decay = "{B0 -> ^K+ ^pi-, B~0 -> ^K- ^pi+}";
```

by explicitly writing out all the decays using braces and respecting the order of the particles such that the particles get the correct labels. The number of flags must be identical in each decay mode.

4.2 MC truth

`DecayChainNTuple` can retrieve the true generated decay and fill in MC truth information. This is done by setting the `MCDecay` and `FillMCDecay` properties:

```
OffBs2JpsiPhiTree.MCDecay = "{
B_s0 ->  (^J/psi(1S) -> ^mu+ ^mu- {, gamma} {, gamma})
         (^phi(1020) -> ^K+ ^K-),
B_s~0 -> (^J/psi(1S) -> ^mu+ ^mu- {, gamma} {, gamma})
         (^phi(1020) -> ^K+ ^K-)}";
OffBs2JpsiPhiTree.FillMCDecay = true;
```

This will look for all the generated true $\mathrm{B_s}$ decays. The syntax used for the decay chain is that of the `MCDecayFinder`. One should keep the same order as in the `Decay` property and the number of particles flagged must be identical to that of the reconstructed decay. The

[11] This is the only thing that cannot be set through options: there would be too many variables to type in. The names of the variables that in some cases are not very explicit may also change.

labelling is the same as for the reconstructed part. Note that the true generated 4-vectors are retrieved from the HepMC format.

The association to the MC truth in DecayChainNTuple is obtained by retrieving a private version of the interface of a DaVinciAssociator tool. The type of associator tool used is Particle2MCLinksAsct with a private name being LinkAsct.[12] The association is done directly on the final states using the Particle2MCLinks algorithm to build the relation table and then by requiring the associated MC particle to correspond to one of the true generated signal particles. The configuration for our example is then:

```
OffBs2JpsiPhiTree.LinkAsct.Location =
                  "Phys/Relations/Particle2MCLinksOffBs2JpsiPhi";
OffBs2JpsiPhiTree.LinkAsct.AlgorithmType = "Particle2MCLinks";
OffBs2JpsiPhiTree.LinkAsct.AlgorithmName =
                  "Particle2MCLinksOffBs2JpsiPhi";
Particle2MCLinksOffBs2JpsiPhi.InputData  =
                              {"Phys/CombinedForBs2JpsiPhi/Particles"};
```

This allows to run several instances of DecayChainNTuple in the same job. The N-tuple variable indicating if a particle is associated to a true signal MC particle is Sig, with the correspond particle's label.

Note that the association is not done for composite particles as the code only looks for direct association: Sig_lab0 will always be zero. However, asking for the selected B_s to be associated to the MC truth is equivalent to requiring all its final states to be associated.

Warning: The correct InputData must be provided to the associator algorithm. This becomes important when using algorithms cloning particles (e.g. FilterDesktop) where the MC association of the original particles can no longer be used.

4.3 Reference

Warning: When setting the different boolean properties to true, make sure you run the necessary code not to run into an exception.

[12] The code uses internally the AssociatorWeighted interface.

DecayChainNTuple

DVAlgorithm — Author: *Luis Fernández*
In Phys/DaVinciMCTools

Algorithm that fills a N-tuple according to a given decay string (default values are for offline use).

Options:

`std::string Decay = "B0 -> ^pi+ ^pi-"`	The reconstructed decay.
`std::string NtupleName = "FILE1/MySelection"`	The name of the N-tuple.
`std::string MCDecay = "B0 -> ^pi+ ^pi-"`	The MC decay.
`bool FillMCDecay = false`	Write the MC part of the N-tuple.
`std::string GeomTool = "GeomDispCalculator"`	Name of the `IGeomDispCalculator`.
`bool RequireTrigger = false`	Write trigger information.
`bool RequireTagging = false`	Write tagging information.

A few more options are available for *online* use:

`bool UseRichOnlinePID = false`	Use of online Rich PID.
`bool UseOnlineCalo = false`	Use of online calorimeter.

Note that `DecayChainNTuple` may also be used in an online HLT environment: using as geometrical tool the `TrgDispCalculator` and by telling the `PVLocator` tool where to look for the reconstructed primaries.

5 Warning

Please note the following warning: GAUDI has not been designed to be "programmed by options". There is no sanity check for options (yet). For instance the following typos would lead to a stop of execution because a tool or algorithm is not found, or because an option is not found:

```
// Typo in MakeResonances:
SeqPreselBd2DPi.Members += {"MakeResonnances/DForPreselBd2DPi" };
// Typo in KinFilterCriterion:
DForPreselBd2DPi.DaughterFilter.Selections = { "K+ : KinfilterCriterion" };
// Typo in MinIPsignif:
DForPreselBd2DPi.DaughterFilter.PVIPFilterCriterion.Minipsignif = 1. ;
```

On the other hand the following typos in instance names will simply be ignored and not produce any warning:

```
SeqPreselBd2DPi.Members += {"MakeResonances/DForPreselBd2DPi" }; // OK.
// Typo in DForPreselBd2DPi -> this line will be ignored.
DForPreselBd2Dpi.DaughterFilter.Selections = {"K+ : KinFilterCriterion"};
// This line is correct:
DForPreselBd2DPi.DaughterFilter.Selections
            += {"pi+  : KinFilterCriterion/PiKin"};
// Typo in PiKin -> the cut will be ignored.
DForPreselBd2DPi.DaughterFilter.piKin.Minipsignif = 1. ;
```

We are well aware that this is not a very user-friendly situation, but we consider it temporary. For the mid-term an "option spell-checker" is under study. For the longer term we expect that the HLT will be steered either by an additional layer of `python` that would generate the options, or directly by a `GaudiPython` script.

6 Conclusion

The development of the HLT has triggered the writing of new generic selection algorithms and tools that can be used both on- and offline. The use of these tools (may it be by configuring everything by options or just by using the filter criteria from a `DVAlgorithm`) ensures a maximal use of similar tools—and hence correlation—of all stages of the selection: HLT, stripping and final selection.

This is a way of maximising total efficiencies by minimising cross-inefficiencies. It also reduces the systematic errors related to the measurements of these cross-inefficiencies.

The present design has been optimised for a quick development of the HLT and we are well aware of its limitations, mainly related to the absence of any sanity check. These tools are bound to evolve with time, as well as the present note.

Appendix

A `MakeResonances` and Photons

The photon being a neutral particle its origin cannot be determined and hence its direction is poorly defined. Photons are therefore *arbitrarily* created at the origin of LHCb's reference frame and pointing to the corresponding electromagnetic calorimeter ECAL clusters allowing in this way to reconstruct the momentum 4-vectors based on the energy of ECAL clusters.[13]

The `UnconstrVertexFitter` has been revisited to prevent the use of photons[14] directly in the vertex fit. Using the `PhotonParams` tool, the photons' parameters are re-evaluated at a reference vertex previously obtained by fitting charged tracks or composite particles.[15]

[13] A direct consequence of this definition is that photons should not be used in a vertex fitter: they should not contribute to the determination of the position of a vertex. However, offline selections used to abuse the fitter by considering the photon as a particle with well-defined direction: the photon is first transported to a given vertex and then fitting all the particles originating from this vertex a composite particle is created. This way of creating composite particles involving photons originates from the fact that the only way of associating particles to a composite is through its vertex.

[14] From now on, what we call photons are `Particles` of the type `ContainedObject`, i.e. they were created from a particle maker. Conversion photons are treated as composite particles.

[15] Note that the original photons are modified by the tool, hence their parameters may differ when retrieved later in the code.

The updated photons are then simply added to the vertex's decay products in order to get the correct invariant mass for the newly formed composite particle.

As a result of this new implementation, the `UnconstrVertexFitter` can no longer be called only with photons (as it used to be the case in previous versions) to create particles only decaying to photons, such as $\eta \to \gamma\gamma$ or $\pi^0 \to \gamma\gamma$. The `UnconstrVertexFitter` now has the following features:

- When fitting only *one* composite particle with photons (e.g. $\mathrm{B_s} \to \phi\gamma$), the vertex is not re-fitted but instead the existing vertex is used to attach the additional photons;[16]
- This fitter is no longer limited to one level of recursiveness but looks for all the descendants to be used for the fit (long-lived particles, resonances' decay products) and updates all the photons' parameters to the resulting vertex;
- New option `bool UseDaughters = true`: by setting this option of the `UnconstrVertexFitter` to `false`, the fitter will ignore all the descendants of the particles to fit.

With these modifications, `MakeResonances` can ignore if the required decay involves photons. However for the special case of particles only decaying to photons, `MakeResonances` can create such particles through the option `MotherToNGammas`.

We give typical examples of decay channels involving photons below.

A.1 Examples of decays involving photons

The simplest example is that of $\mathrm{B_s} \to \phi\gamma$. The configuration of `MakeResonances` is identical to the case without photons. If `Bs2PhiGamma` is an instance of `MakeResonances`, then to get all the $\phi\gamma$ combinations one just needs the following option:

```
Bs2PhiGamma.DecayDescriptor = "B_s0 -> phi(1020) gamma";
```

`MakeResonances` can also create composite particles only decaying to photons, such as $\eta \to \gamma\gamma$. This is done using the `MotherToNGammas` property. The code will combine the required number of photons, create the resulting composite particle whose vertex is set at the origin of the reference frame. The photons parameters are not re-evaluated since the mother particle is created at the origin, but they are just added to the mother's vertex.

An example of use is illustrated in the selection of $\mathrm{B_s} \to \mathrm{J}/\psi\eta(\gamma\gamma)$. The η is created with:

```
ApplicationMgr.TopAlg += {"MakeResonances/Eta2GGForBs2JpsiEta2GammaGamma"};
Eta2GGForBs2JpsiEta2GammaGamma.PhysDesktop.InputLocations =
                                          {"Phys/PhotonsForBs2JpsiEta2GG"};
Eta2GGForBs2JpsiEta2GammaGamma.DecayDescriptor = "eta -> gamma gamma";
Eta2GGForBs2JpsiEta2GammaGamma.Window = 60.0 * MeV;
Eta2GGForBs2JpsiEta2GammaGamma.MotherToNGammas = true;
```

with a mass cut of ± 60 MeV applied around the η nominal mass. One gets the final $\mathrm{B_s}$ candidates by combining these η with existing J/ψ in a different instance `Bs2JpsiEta2GammaGamma` of `MakeResonances`:

[16] This is also an abuse of the vertex fitter that should disappear, but needs a modification of the physics event model.

```
Bs2JpsiEta2GammaGamma.DecayDescriptor = "B_s0 -> J/psi(1S) eta";
```

where the vertex of the B_s is actually the J/ψ one, vertex at which the photons from the η are re-evaluated in order to get the correct B_s momentum-vector. Note that `MakeResonances` does not re-compute the η momentum components and covariance matrix with the displaced photons and its vertex remains at the origin. It is up to the user in a private analysis algorithm to retrieve the selected candidates and take corrections properly into account. In this example, the decay tree will look like:

Name	E	M	P	Vz
	MeV	MeV	MeV	mm
B_s0	56199.36	5373.15	55941.91	2.79
+-->J/psi(1S)	37596.04	3095.27	37468.41	2.79
|+-->mu+	14235.50	105.66	14235.11	3.50
|+-->mu-	23360.54	105.66	23360.30	2.05
+-->eta	18603.32	543.36	18595.39	0.00
+-->gamma	11731.91	0.00	11731.91	2.79
+-->gamma	6871.42	-0.00	6871.42	2.79

References

[1] P. KOPPENBURG AND L. FERNÁNDEZ. HLT Exclusive Selections. LHCb-2005-015, LPHE-2005-011, upcoming, 2005.

[2] MURIEL PIVK AND FRANCOIS R. LE DIBERDER. sPlot: a statistical tool to unfold data distributions. *Nucl. Instrum. Meth.*, A555:356–369, 2005.

[3] IVAN BELYAEV. LoKi: Smart & Friendly C++ Physics Analysis Toolkit. LHCb-2004-023, 2004.

[4] HUGO RUIZ. Guidelines for selection algorithms. LHCb-2004-031, 2004.