Table of Contents

DaVinci Tutorial 4

Introduction

The purpose of this exercise is to allow you to write a set of complete though simple selection algorithms for a typical decay: B_s J/ . We will use Configurables and the Particle Selection Framework to write re-usable J/ , and B_s Selections and obtain a algorithm sequence that can be passed on to DaVinci to run the selection.

Slides

This tutorial corresponds to the slides shown at the 2009 January LHCb-UK tutorial here and the 2011 Course here.

Warning: This tutorial refers to DaVinci version v30r2 or later.

Prerequisites

This tutorial requires you to have done DaVinciTutorial0, DaVinciTutorial0p5 and all steps leading up to that. At this stage you should have an options file and Ganga job working with DaVinci.

At this stage you will need to know quite a bit more about python, including what a python module is, how you create and import them, and the nature of the PYTHONPATH.

This tutorial assumes that you have run

```
SetupProject DaVinci --build-env
getpack Tutorial/Analysis
```

for your personal Davinci installation. Any reference to the folder `solutions` can be found in the `Tutorial/Analysis` folder

1 LoKi

Our CombineParticles framework uses the amazing flexibility of LoKi. If you want to know more you should follow the LoKiTutorial .

However, this is not a prerequisite.

1.1 LoKi cuts

We will use LoKiHybridFilters to apply cuts. They are python objects hiding some C++ code behind them. You can play with them in options as in python. Try for instance the following, outside ganga for the moment:

SetupProject DaVinci
python -i

```
from math import sqrt
from LoKiPhys.decorators import *
from LoKiCore.functions import monitor
p = LHCb.Particle()
p.setParticleID( LHCb.ParticleID(11) )
m = p.momentum()
m.SetPx (   1000 )
```

```
m.SetPy (  -1000 )
m.SetPz (  10000 )
m.SetE  ( sqrt( m.P2() + 5000*5000 ) )
p.setMomentum ( m )
fun = PX+PY
print PX(p), PY(p) , fun(p)
fun2 = PX>750
print fun2(p)
fun3 = monitor(fun2)
print fun3(p)
from LoKiCore.doxygenurl import browse
browse(PT)
```

2 Particle Selection framework

We have a powerful python module-based selection framework. This is the recommended way of constructing particles. See, for example, the options files in $DAVINCIROOT/options for more examples.

Firstly we will use the GaudiAlgorithm CombineParticles to create our resonances, then we will use the framework to combine all these into a sequence. From Following DaVinciTutorial0 you will have a job which runs inside Ganga. You should copy that options file

We will be coding the , J/ and B_s selections as self-contained Selection configurables within python modules. This means that, besides using them for this example, we will be able to use them to build other particles, to make (Misro)DSTs, even to bolt them seamlessly into the stripping. For this we will use the Particle Selection "Framework". We will be writing python modules, so your solution should be put in python/DaVinci4/Bs2JpsiPhi.py. This will define a python module, which you will be able to import like this:

```
from DaVinci4 import Bs2JpsiPhi
help(Bs2JpsiPhi)
dir(Bs2JpsiPhi)
```

2.1 Make a python module

We need to create a python module that will end up in the `$PYTHONPATH`. This package is set up such that anything in python/DaVinci4 gets put in an area that is on that path once the package is built.

- Open empty file python/DaVinci4/Bs2JpsiPhi.py
- Add some necessary modules

```
import GaudiKernel.SystemOfUnits as Units
from Gaudi.Configuration import *
```

- Build the package and check the module is in the InstallArea (which is in the $PYTHONPATH)

In your ganga session

```
t.application.make()
env=t.application.getenv()
env['PYTHONPATH']
```

2.2 Make a loose J/ Selection

Add this into your python file

```
# make a Selection out of a data-on-demand location
from Configurables import CombineParticles
_jpsi2mumu = CombineParticles("Jpsi2MuMu")
```

```
# Define the Decay Descriptor
_jpsi2mumu.DecayDescriptor = "J/psi(1S) -> mu+ mu-" # mandatory
```

2.3

CombineParticles is a GaudiAlgorithm seemlessly integrating into LoKi, this algorithm creates a mother particle with the correct four-vector and endvertex to have decayed into the particles we saw in the detector.

Unfortunately we have a lot of 'background' events, particles from the primary vertex or from decays of other particles, or perhaps particles that are not really muons at all. Hence we 'cut' away at the background to see the signal.

- There are three cuts applied in `CombineParticles`
    1. `DaughtersCuts` : On the incoming daughter particles. Note the double plural.
    2. `CombinationCut` : Once a combination has been made according to the decay descriptor but before the vertex fit. This cut is applied to the array of particles to be used in the vertex fit.
    3. `MotherCut` : On the outcoming Mother, after the vertex fit.

All the cuts that require the position of the vertex must be applied in `MotherCut`, while the others, like mass, can be applied earlier (saving CPU). Note that for long lived particles like Ks it pays off to apply a loose mass cut in `CombinationCut` and a harder in `MotherCut`. The reason is that the vertex fit does a propagation of the momenta through the detector. You thus get the momentum at the Ks vertex, while in `CombinationCut` it's just the sum of the momenta of the daughters at their first measurement.

- The cuts that can be applied in the filters are the LoKiHybridFilters.
- Let's ignore cuts on daughters for the time being.
- We want a mass cut on the J/ candidates.

```
_jpsi2mumu.CombinationCut = "ADAMASS('J/psi(1S)')<30*MeV"
```

- Note that here we use functors for an *array* of particles. Usually there's an "A" in the name to remind you about this. `ADAMASS` is the array version of `ADMASS`. See LoKiHybridFilters for a complete list.
- Let's also cut on the vertex chi2. This happens after the vertex fit and is therefore a Mother cut.

```
_jpsi2mumu.MotherCut = "(VFASPF(VCHI2/VDOF)<10)"
```

2.4 Add into your selection

- Now we make a Selection object for the Jpsi. It only needs the standard loose muons, so we pass it a list with `_muons` as only entry for `RequiredSelections`.

```
from PhysSelPython.Wrappers import Selection
from StandardParticles import StdLooseMuons
LooseJpsi2MuMu = Selection("SelLooseJpsi2MuMu",
                            Algorithm = _jpsi2mumu,
                            RequiredSelections = [StdLooseMuons])
```

Note that we do not need to specify any `Inputs` for the `CombineParticles`. The `Selection` takes care of setting it from the `RequiredSelections`. With this we are already at the same level as with our C++ code for the J/ ! In fact, we are beyond that level, because now we can use the Jpsi2MuMu selection from python:

```
from DaVinci4.Bs2JpsiPhiEx4 import LooseJpsi2MuMu
```

3 Creating the full sequence

3.1 Filter further to make a tighter J/ Selection

We can apply harder cuts using a `FilterDesktop`.

```
# make a FilterDesktop
from Configurables import FilterDesktop
_jpsifilter = FilterDesktop("_JpsiFilter",
                            Code = "(PT>1*GeV) & (P>3*GeV)")
#make a Selection out of it
Jpsi2MuMu = Selection("SelJpsi2MuMu",
                       Algorithm = _jpsifilter,
                       RequiredSelections = [LooseJpsi2MuMu])
```

This applies a momentum and transverse momentum cut on the J/ . Of course that could have been done directly in the `CombineParticles` above. Again, note that Jpsi2MuMu doesn't need to explicitly know its `Inputs`. It only needs access to LooseJpsi2MuMu, which could've even been in a different module in a different package.

- Note that we are not doing logical operations but binary additions. Hence use `&` and `|`, not `&&`, `||`, `and`, `or`.

3.2 Make the (1020) Selection

Now make another `Selection` called `Phi2KK`, using a `CombineParticles` instance for `"phi(1020) -> K+ K-"`. Apply a 50 MeV mass cut and a vertex chi2 less than 100. The input should come from `StdLooseKaons` which you need to pass to `Phi2KK` as `RequiredSelections`.

3.3 Make the Bs Selection

The last step is the Bs using the filtered J/ and the as input. A good cut for a B is to require pointing to the PV. This needs the vertex and thus is a `MotherCut` cut. Let's also cut on the vertex chi2. We also want a mass cut, but let's make it wide so we get an idea of our background level. ± 500 MeV is typical in the HLT and stripping, but here let's make it 2 GeV.

```
_bs2jpsiphi = CombineParticles("Bs2JpsiPhi",
             DecayDescriptor = "B_s0 -> phi(1020) J/psi(1S)",
             CombinationCut = "ADAMASS('B_s0')<2*GeV",
             MotherCut = "(VFASPF(VCHI2/VDOF)<10) & (BPVIPCHI2()<100)" )
# feel free to add more cuts
_bs2jpsiPhi. ....
# Now let's add some plots
from Configurables import LoKi__Hybrid__PlotTool as PlotTool
_bs2jpsiphi.HistoProduce = True
_bs2jpsiphi.addTool( PlotTool("DaughtersPlots") )
# Note that it's using the same functors as above. Hence the same syntax.
_bs2jpsiphi.DaughtersPlots.Histos = { "P/1000"  : ('momentum',0,100) ,
                                      "PT/1000" : ('pt_%1%',0,5,500) ,
                                      "M"       : ('mass in MeV_%1%_%2%_%3%',0.8*Units.GeV,4*Units
_bs2jpsiphi.addTool( PlotTool("MotherPlots") )
_bs2jpsiphi.MotherPlots.Histos = { "P/1000"  : ('momentum',0,100) ,
                                   "PT/1000" : ('pt_%1%',0,5,500) ,
                                   "M"       : ('mass in MeV_%1%_%2%_%3%',4*Units.GeV,6*Units.GeV)
# now make the Selection
Bs2JpsiPhi = Selection("SelBs2JpsiPhi",
                    Algorithm = _bs2jpsiphi,
                    RequiredSelections = [ Jpsi2MuMu, Phi2KK ])
```

Make sure the `Selection` definition comes **after** the complete definition of all algorithms that go inside.

3.4 Make a B_s J/ (μμ) (KK) SelectionSequence

Now that we have the J/ , and B_s Selections (plus Selections for the data-on-demand particles) we can create a `SelectionSequence` for the B_s J/ (μμ) (KK)selection. It is remarkable simple, since all the necessary information is contained in each Selection.

```
from PhysSelPython.Wrappers import SelectionSequence
SeqBs2JpsiPhi = SelectionSequence('SeqB2jJpsiPhi', TopSelection = Bs2JpsiPhi)
```

4 Run!

4.1 Build!

In your ganga session

```
t.application.make()
```

4.2 Run

Add into your options file:

```
from DaVinci4.Bs2JpsiPhiEx4 import SeqBs2JpsiPhi as theSequence
MySelection = theSequence.sequence()
DaVinci().UserAlgorithms = [MySelection]
```

Then submit a new job in Ganga.

Help!

The solution is given in `python/DaVinci4/solutions` for the module. and `solutions/DaVinci4` for the script to run the selection.

What next?

- Look at the examples in the ParticleSelection if you want to know more about the ParticleSelection framework
- To know how to monitor what you just did go to DaVinciTutorial5
- To know how to do the same thing in C++ and learn about writing DaVinci code go to DaVinciTutorial1 and then do 2 and 3.

-- PatrickKoppenburg - 01 Oct 2007 -- PatrickKoppenburg - 13 Jun 2008 -- PatrickKoppenburg - 05 Jan 2009 -- JuanPalacios - 2009-10-02 -- PatrickSKoppenburg - 16-Oct-2012 -- PatrickSKoppenburg - 30-Sep-2013

--- This topic: LHCb > DaVinciTutorial4
Topic revision: r43 - 2014-07-29 - NathanaelFarley