Table of Contents

DaVinci Tutorial 8

How to run on a stripped DST.

This has been last shown on 24/10/2012. See the slides.

Prerequisites

This assumes you know how to tun DaVinci and know a bit about DecayTreeTuple. Do DaVinciTutorial0 and DaVinciTutorial0p5 before. Have a look at (or do) DaVinciTutorial6.

Think!

Before you start analysing anything you should think! We assume you know

- Which stripping lines select your candidates
- Which trigger lines select your candidates

"I'll take all" is not a valid answer. Talk to your supervisor.

Foreword

What is the Stripping? : Stripping is the centralized selection of interesting events run after the reconstruction. Stripped DSTs, stripped events, and stripping selections are the only selections which are run centrally, and therefore the only events and selections which are available for the entire LHCb dataset.

What is a StrippingLine? : A line is the sequence of selections used to create candidates and select the event

Where are the lines and cuts? : Stripping Selections live in the package Phys/StrippingSelections, they usually hard-code some fiducial cuts and soft-code the majority of cuts which are defined at run-time by passing a dictionary into each `LineBuilder` when it is constructed in python. The `LineBuilder` is a simple python object that knows how to construct a set of Stripping Lines. The cuts used in the strippings are stored in a database, like a TCK, but for the stripping which lives in the Phys/StrippingSettings package. Since quite often lines are changed in non-backward compatible ways, removed or added, the Phys/StrippingArchive package stores the python classes actually used in the stripping from Stripping13 onwards.

What is a Stream? : A stream is a group of selections of a similar type, selecting similar events. Ideally no analysis should have to work with the output of more than one stream. Streams usually have a corresponding working group which approves the lines within it. Each stream writes out a different DST with the selected events and candidates from that stream.

Tools and help

This documentation in the LHCb release area lists all the stripping20 streams and lines and gives a complete (and somewhat cryptic) list of the filters used to define the lines.

The StrippingSettings and StrippingArchive python packages have tools that give the lines and configurations used by the strippings.

Open a new shell and call `SetupProject DaVinci` directly:

```
SetupProject DaVinci v33r7
python
from StrippingArchive import Utils
help(Utils)
from StrippingSettings import Utils as SettingUtils
help(SettingUtils)
```

If the `DaVinci` version is not available try

```
SetupProject DaVinci --list-versions
```

Finding out about a stream

The most recent stripping is Stripping20 (Stripping13, 14, etc. contain older definitions of the stripping lines). The following

```
print Utils.streamNames("stripping20")
```

prints a list of the StrippingStreams that appear in Stripping20, including, for example, "Bhadron", "Semileptonic", "Dimuon", etc.

Finding out about a line

Each stream contains different line builders used to construct stripping lines. The following

```
SettingUtils.printBuildersByStream("stripping20")
```

will print a list of the names of the builders appearing in Stripping20, grouped by the stream they appear in. Meanwhile,

```
SettingUtils.printBuildersByWG("stripping20")
```

will print the same information, but this time grouped by the working group (WG) in charge of the builder.

For example, one of the builders belonging to the "Semileptonic stream" is `B0q2DplusMuX`. We can examine this line builder using the `lineBuilderAndConf` method of the `StrippingArchive.Utils` module:

```
line,conf=Utils.lineBuilderAndConf("stripping20","B0q2DplusMuX")
help(line)
print conf
```

The object `conf` is the python dictionary that was used to configure the `LineBuilder` for the `B0q2DplusMuX` stripping lines in Stripping20, while `line` is a copy of the line builder. The name `B0q2DplusMuX` is important. It's the name the `LineBuilder` was associated with in the Stripping. This name is usually (but not always) the name that appears after `Stripping' in the filename containing the `LineBuilder` in StrippingSelections.

Where is the line stored in the stripping?

When the analysis application is run it will be passed the location of the stripping data (a collection of .DST files) to be processed; this output location is the location of the line inside the stripping data store. The output location in the stripping can be found using

```
print Utils.outputLocations("stripping20","B0q2DplusMuX")
```

As you can see, there are several output locations. The `LineBuilder` `B0q2DplusMuX` was used to produce several related lines in the stripping, as is common.

More information can also be obtained using the `STCKsh` executable= (after running `SetupDaVinci`).

Generic workflow for a particle reconstruction analysis

1. Read your candidates
2. Refine your candidates (more cuts)
3. TisTos your candidate
4. Store (or fit) it somehow. That could be done directly in python, or via a microDST, or in DecayTreeTuple. For simplicity, we'll use this option.

J/ μμ analysis

Where is the candidate?

We can, logically, look for the J/ to dimuon decay in the Dimuon stream. Examining the list of dimuon streams in this documentation, we find that there are two lines `StrippingBetaSJPsi2MuMuDetachedLine` and `StrippingBetaSJPsi2MuMuLine`, the former of which has some tighter cuts (check this). By looking at the builders for the Dimuon stream (see above), clearly this line belongs was created by the BetaS builder. So, we can find the output location for the line using

```
SetupProject DaVinci  v33r7
python
from StrippingArchive import Utils
print Utils.outputLocations("stripping20","BetaS")
```

This prints out the locations for all the lines created by the BetaS builder. Looking through it, the location of the `StrippingBetaSJPsi2MuMuDetachedLine` line is `/Event/Dimuon/Phys/BetaSJpsi2MuMuDetachedLine/Particles`.

Is this really the line I want?

It is, or the tutorial wouldn't have told you so. But let's pretend you're not sure. You can check that with `PrintDecayTree`. There is more information on this tool in DaVinciTutorial5. Essentially, this algorithm will print the reconstructed decay tree for reconstructed events.

Create the following python options file

```
from Gaudi.Configuration import *
line = "BetaSJpsi2MuMuDetachedLine"
location = "/Event/Dimuon/Phys/"+line+"/Particles"
from Configurables import DaVinci, PrintDecayTree
pt = PrintDecayTree(Inputs = [ location ])
DaVinci().appendToMainSequence( [ pt ] )
DaVinci().DataType = "2012"
DaVinci().EvtMax = -1
```

You can run this DaVinci job in any of the possible ways described so far (through `ganga`, `gaudirun.py`, or `cmt run gaudirun.py`). The output will include `PrintDecayTree` output that will look something like

```
PrintDecayTree....   INFO
<---------------------------------- Particle ----------------------------------->
           Name          E          M          P         Pt        phi         Vz
                       MeV        MeV        MeV        MeV       mrad         mm
J/psi(1S)             67092.32   3096.92   67020.81  17339.45  -2095.73    -29.40
+-->mu+               36447.87    105.66   36447.71   8322.81  -1978.59    -41.79
+-->mu-               30703.64    105.66   30703.46   9140.31  -2202.21    -38.35
```

But Tutorial 8, not all the events have candidates!

Yes, that is true. The configurable parameter `DaVinci().EvtMax` sets how many events to look at in the .DST file. Look again at the logical location of the stripping line in the stripping file:

```
/Event/Dimuon/Phys/BetaSJpsi2MuMuDetachedLine/Particles
```

The stripping file is logically sorted event first, rather than line first. Not every event passes the filter for every line. The code above this still calls `PrintDecayTree` for every event processed, which leads to a performance penalty. This penalty is small, but we can introduce a couple of important tools while showing how to avoid this.

The first of these tools is `GaudiSequencer`. This tool lets you create a sequence of algorithms which halts after one of the algorithms fails (read more in its documentation). For the first algorithm, you can select events which pass the stripping line using an `HltDecReports` filter. Then you can add `PrintDecayTree` as the second algorithm. Then `PrintDecayTree` will run only on events that pass the appropriate filter.

```
from Gaudi.Configuration import *
line = "BetaSJpsi2MuMuDetachedLine"
location = "/Event/Dimuon/Phys/"+line+"/Particles"
from Configurables import DaVinci, PrintDecayTree, GaudiSequencer
from Configurables import LoKi__HDRFilter
MySequencer = GaudiSequencer('Sequence')
pt = PrintDecayTree(Inputs = [ location ])
sf = LoKi__HDRFilter( 'StripPassFilter', Code="HLT_PASS('Stripping"+line+"Decision')", Location="
MySequencer.Members = [ sf, pt ]
DaVinci().appendToMainSequence( [ MySequencer ] )
DaVinci().DataType = "2012"
DaVinci().EvtMax = -1
```

The output will now include a counter named `StripPassFilter` which counts how many events passed this filter.

What about trigger lines?

The same can be done to count how many of the events pass a high level trigger. Add the following code before `MySequencer` is appended to the main sequence:

```
tf = LoKi__HDRFilter( 'HltPassFilter', Code="HLT_PASS('Hlt1.*Muon.*Decision')" )
MySequencer.Members += [ tf ]
```

This will add a counter which counts how many of the events that pass the stripping line decision also pass these high level muon triggers. To really understand what the trigger does to your signal, though, you should use the TriggerTisTos tools. See also HltEfficiency.

How to refine candidates

To refine candidates, the important point is to **start from your candidate** and apply further cuts. The main tool for this task is the ParticleSelection framework (follow the link for more information). The following is a simple example that applies a mass cut to the J/ candidates:

```
# get classes to build the SelectionSequence
from PhysSelPython.Wrappers import AutomaticData, Selection, SelectionSequence
# Get the Candidates from the DST. AutomaticData is for data on the DST
JpsiSel = AutomaticData(Location = location)
# Filter the Candidate.  Let's throw away everything above 4 GeV
from Configurables import FilterDesktop
_jpsiFilter = FilterDesktop('jpsiFilter', Code = '(M>2500*MeV) & (M<4000*MeV)')
```

```
# make a Selection
JpsiFilterSel = Selection(name = 'JpsiFilterSel',
                          Algorithm = _jpsiFilter,
                          RequiredSelections = [ JpsiSel ])

# build the SelectionSequence
JpsiSeq = SelectionSequence('SeqJpsi',
                            TopSelection = JpsiFilterSel,
                           )

DaVinci().appendToMainSequence( [ JpsiSeq.sequence() ] )
```

How to write particle information into a ROOT tuple

This task is the focus of DaVinciTutorial6, and for more information look there. The following code snippet writes typical interesting variables into a ROOT file named `Tutorial8.root`.

```
from Configurables import DecayTreeTuple, TupleToolTrigger, TupleToolDecay, TupleToolTISTOS
tuple = DecayTreeTuple("Jpsi_Tuple")
# tuple.addTupleTool( "TupleToolGeometry") // already default
# tuple.addTupleTool( "TupleToolKinematic")// already default
tuple.addTupleTool( "TupleToolPropertime")
tuple.addTupleTool( "TupleToolPrimaries")
# tuple.addTupleTool( "TupleToolEventInfo")// already default
tuple.addTupleTool( "TupleToolTrackInfo")
tuple.Decay = "J/psi(1S) -> ^mu+ ^mu-"
tuple.Inputs = [ JpsiSeq.outputLocation() ]
tuple.addTupleTool(TupleToolTISTOS)
tuple.TupleToolTISTOS.TriggerList = [ "Hlt2DiMuonJPsiDecision", "Hlt2DiMuonJPsiHighPTDecision", "
tuple.TupleToolTISTOS.VerboseHlt2 = True
DaVinci().appendToMainSequence( [ JpsiSeq.sequence(), tuple ] )
DaVinci().appendToMainSequence( [ tuple ] )
DaVinci().TupleFile = "Tutorial8.root"
```

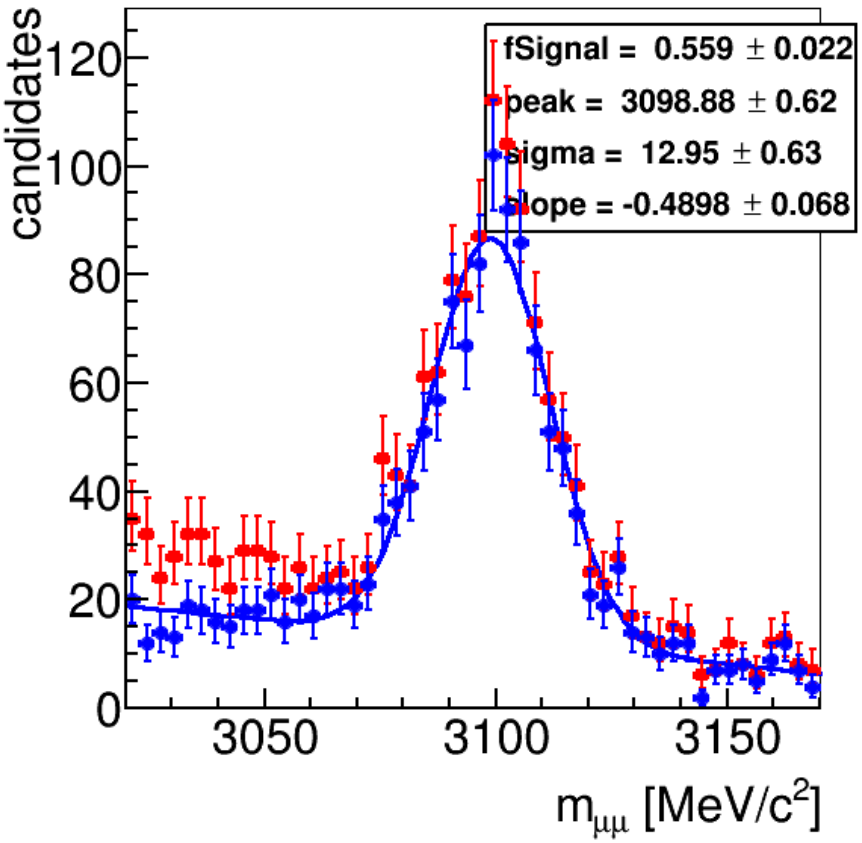
Putting it all together

The following shows how put all these tasks together into an analysis sequence. This is taken (with a few modifications) from a solution options file available in `Tutorial/Analysis/solutions/DaVinci8` from tag `v10r6`.

```
##############################################################################
#
# Options for exercise 8
#
# @author Patrick Koppenburg
# @date 2010-06-07
#
##############################################################################

from Gaudi.Configuration import *
from Configurables import DaVinci

######### Candidate location
line = 'BetaSJpsi2MuMuDetachedLine'
location = '/Event/Dimuon/Phys/'+line+'/Particles'

######### Refining the candidate
# get classes to build the SelectionSequence
from PhysSelPython.Wrappers import AutomaticData, Selection, SelectionSequence
# Get the Candidates from the DST. AutomaticData is for data on the DST
JpsiSel = AutomaticData(Location = location)
# Filter the Candidate.  Let's throw away everything above 4 GeV
```

```
from Configurables import FilterDesktop
_jpsiFilter = FilterDesktop('jpsiFilter', Code = '(M>2500*MeV) & (M<4000*MeV)')

# make a Selection
JpsiFilterSel = Selection(name = 'JpsiFilterSel',
                          Algorithm = _jpsiFilter,
                          RequiredSelections = [ JpsiSel ])

# build the SelectionSequence
JpsiSeq = SelectionSequence('SeqJpsi',
                            TopSelection = JpsiFilterSel,
                           )
DaVinci().appendToMainSequence( [ JpsiSeq.sequence() ] )

############ DecayTreeTuple
from Configurables import DecayTreeTuple, TupleToolTrigger, TupleToolDecay, TupleToolTISTOS
tuple = DecayTreeTuple("Jpsi_Tuple")
tuple.ToolList +=  [
    "TupleToolGeometry"
    , "TupleToolKinematic"
    , "TupleToolPrimaries"
    , "TupleToolEventInfo"
    , "TupleToolTrackInfo"
    , "TupleToolTISTOS"
    , "TupleToolAngles"
    , "TupleToolPid"
    , "TupleToolPropertime"
    ]
tuple.Decay = "J/psi(1S) -> ^mu+ ^mu-"
tuple.Inputs = [ JpsiSeq.outputLocation() ]
tuple.addTool(TupleToolTISTOS)
tuple.TupleToolTISTOS.TriggerList = [ "Hlt2DiMuonJPsiDecision" ]
tuple.TupleToolTISTOS.VerboseHlt2 = True
DaVinci().appendToMainSequence( [ tuple ] )
DaVinci().TupleFile = "Tutorial8.root"
##################

######### Debugging
from Configurables import GaudiSequencer
MySequencer = GaudiSequencer('Sequence')

# decision filter test
from Configurables import LoKi__HDRFilter
sf = LoKi__HDRFilter( 'StripPassFilter', Code="HLT_PASS('Stripping"+line+"Decision')", Location="
MySequencer.Members = [ sf ]

# TES explorer
# from Configurables import StoreExplorerAlg
# MySequencer.Members += [ StoreExplorerAlg(PrintEvt=100) ]

# Print decay tree
# from Configurables import PrintDecayTree
# MySequencer.Members += [ PrintDecayTree( 'PrintDiMuons', Inputs = [ location ] ) ]

DaVinci().appendToMainSequence( [ MySequencer ] )

######### Options
DaVinci().DataType = "2012"
DaVinci().EvtMax = 100000
DaVinci().PrintFreq = 100
```

Opening the ROOT file (which can be done from within `ganga` using `peek('Tutorial8.root')`), you can produce examine the histograms saved by the tuple tools included in the above script. In particular, the J/ mass should look something like this:



Here the red points are all candidates passing the selection and the clue ones only those that are `Hlt2DiMuonDetachedJPsi` TOS.

Next

Go back to any of the other DaVinciTutorial s, then clean up and get ready for your new analysis following: DaVinciTutorialTips.

-- PatrickSKoppenburg - 08-Jun-2010 -- PatrickSKoppenburg - 16-Oct-2012 -- PatrickSKoppenburg - 30-Sep-2013

--- This topic: LHCb > DaVinciTutorial8
Topic revision: r57 - 2014-11-04 - GiulioDujany