Table of Contents

DaVinci Tutorial 1

We start with a very simple algorithm that looks at muons and fills some histograms. Follow it step-by-step. Every single line you need to write is given, but be very careful about where to paste what!

Slides

This tutorial refers to the slides shown here.

Prerequisites

DaVinciTutorial0 and DaVinciTutorial0p5

Start to write the options

- First make sure you have the LHCb flavour of emacs. If not do what's described here.
- Open the main options file (from ganga type:)

```
!emacs $t.application.optsfile[0].name &
```

or from outside ganga move to the options directory of the Tutorial/Analysis package and type:

```
emacs myOptions.py &
```

- Write in the options file.

```
from Gaudi.Configuration import *
#
# 1) Let's define a sequence
#
from Configurables import GaudiSequencer
tutorialseq = GaudiSequencer("TutorialSeq")
#
# 2) Create the Tutorial Algorithm
#
from Configurables import TutorialAlgorithm
tutalg = TutorialAlgorithm()
tutorialseq.Members += [ tutalg ]
tutalg.Inputs = [ "Phys/StdAllLooseMuons/Particles" ]
#
# 3) Configure the application. We'll come back to this later.
#
from Configurables import DaVinci
DaVinci().UserAlgorithms = [ tutorialseq ]
```

Since we want to make a J/psi->mumu we only need muons as input Particles. The `StdAllLooseMuons` are all charged tracks compatible with being a muon. They are made on demand when you need them.

Now we need to write the algorithm!

Start to write the algorithm

We will make a small algorithm that loops over muons and plots some variables.

- Open the files for your algorithm (from the shell type):

```
emacs ~/cmtuser/DaVinci_v33r7/Tutorial/Analysis/src/TutorialAlgorithm.{cpp,h} &
```

Answer "D" for `DaVinciAlgorithm` twice. You can also do that from the shell prompt by opening another window. This has created a template for a `TutorialAlgorithm` inheriting from `DaVinciAlgorithm`. But as we will want to use a few more features, replace all instances of `DaVinciAlgorithm` by `DaVinciTupleAlgorithm`. This allows to get access to helper methods to make plots. (For what we plan to do, `DaVinciHistoAlgorithm` would work too).

- Create a new private method in the header file:

```
private:
  StatusCode loopOnMuons(const LHCb::Particle::ConstVector&)const ;
```

- Now let's add something to the execute method in the cpp:

```
StatusCode TutorialAlgorithm::execute() {
  if (msgLevel(MSG::DEBUG)) debug() << "==> Execute" << endmsg;
  StatusCode sc = StatusCode::SUCCESS ;

  // code goes here
  const LHCb::Particle::ConstVector  muons = particles();
  sc = loopOnMuons(muons);
  if (!sc) return sc;

  setFilterPassed(true);  // Set to true if event is accepted.
  return StatusCode::SUCCESS;
}
```

Here we get the particles from a local container and call our new method.

- Now implement the method in the cpp file. You can get a template method typing `insert M` in `emacs`. You then have to replace the type `void` by `StatusCode` and declare the arguments:

```
//=============================================================
// loop on muons
//=============================================================
StatusCode TutorialAlgorithm::loopOnMuons( const LHCb::Particle::ConstVector& muons)const {
  StatusCode sc = StatusCode::SUCCESS ;

  // code goes here

  return sc ;
}
```

- In the method add:

```
for ( LHCb::Particle::ConstVector::const_iterator im = muons.begin() ; im != muons.end() ; ++im )
   plot((*im)->p(), "P", "Muon P", 0., 50.*Gaudi::Units::GeV); // momentum
   plot((*im)->pt(), "Pt", "Muon Pt", 0., 5.*Gaudi::Units::GeV ); // Pt
   debug() << "Mu Momentum: " << (*im)->momentum() << endmsg ;
  }
```

- Now get the primary vertices (**before** the muons loop):

```
const LHCb::RecVertex::Range pvs = this->primaryVertices();
```

- And loop over them (**inside** the muons loop!):

```
for ( LHCb::RecVertex::ConstVector::const_iterator ipv = pvs.begin() ;
          ipv != pvs.end() ; ++ipv ){
      double IP, IPchi2;
      if (msgLevel(MSG::DEBUG)) debug() << (*ipv)->position() << endmsg ;
```

```
      sc = distanceCalculator()->distance((*im), (*ipv), IP, IPchi2);
      if (sc){
        plot(IP, "IP", "Muon IP", 0., 10.*Gaudi::Units::mm);
        plot(IPchi2, "IPchi2", "Muon chi2 IP", 0., 25.);
        if ( (*im)->pt()>2*Gaudi::Units::GeV)
          plot(IP, "IP_2", "Muon IP for PT>2GeV", 0., 10.*Gaudi::Units::mm);
      }
    }
```

- Now compile (from ganga):

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

or from outside ganga, SetupProject --build-env DaVinci v33r7 (if not already done), move to the cmt directory of Tutorial/Analysis and type:

```
cmt make
```

... and fix all compilation errors...

Run!

Submit from within ganga:

- Define a job. If you want the output to the terminal do:

```
j = Job( t, backend = Interactive() )
```

you can also use `LSF`, `Local`, `Dirac`... For small tests, interactive or local is usually best. For larger amounts of data, you'll need to run either on the LSF batch system or on the Grid.

- Before you submit you can export the file:

```
export (j , "Tutorial.py")
```

- Submit the job

```
j.submit()
```

Run it interactively (outside ganga):

Alternatively you can run interactively on `lxplus`. To do so you must use the PFNs. Don't forget to do SetupProject if needed:

```
SetupProject DaVinci v33r7
```

then type:

```
gaudirun.py options/myOptions.py options/Bs2JpsiPhi_Sim08a.py
```

Look at the histograms

If you ran interactively, from outside ganga, then the DVHistos.root file should be in the directory where you ran DaVinci. From that directory, start root by typing:

```
root -l
```

If you ran using ganga, the output of the job will be stored in `j.outputdir`. Go to that directory in ganga, and

you can start root directly:

```
cd $j.outputdir
root -l
```

Or use ganga's built-in peek command:

```
j.peek('DVHistos.root', 'root -l')
```

Or, if you have screwed up your own root environment you may need to exit to the shell

```
SetupProject Gaudi ROOT

root.exe
```

Once in root:

The simplest way to look at histograms is to use the `TBrowser`.

```
TBrowser B
```

which has a Windows-like look and feel.

- But if you just want to save the histograms, you could as well just paste the following in the root prompt

```
TCanvas* c1 = new TCanvas("c1","Tutorial",800,800);
c1->SetLogy();
TFile* F = new TFile("DVHistos_1.root")
```

and make sure you give the correct path for your file! And then

```
F->ls()
F->cd("TutorialAlgorithm")
F->ls()
TH1D* H1 = F->Get("TutorialAlgorithm/P")
TH1D* H2 = F->Get("TutorialAlgorithm/Pt")
TH1D* H3 = F->Get("TutorialAlgorithm/IP")
TH1D* H4 = F->Get("TutorialAlgorithm/IPchi2")

H1->SetLineColor(2)
H1->SetLineWidth(3)
H1->Draw()
gPad->SaveAs("DV1_P.png");

H2->SetLineColor(2)
H2->SetLineWidth(3)
H2->Draw()
gPad->SaveAs("DV1_Pt.png");

H3->SetLineColor(2)
H3->SetLineWidth(3)
H3->Draw()
gPad->SaveAs("DV1_IP.png");

H4->SetLineColor(2)
H4->SetLineWidth(3)
H4->Draw()
gPad->SaveAs("DV1_IPchi2.png");
```

For sure one could do this shorter...

- To quit root:

```
.q
```

Once in root:

Next

- If you want to learn how to write an algorithm doing J/psi->mumu in C++ go to DaVinciTutorial2. (Advanced).
- If you want to learn how to use existing options files, and existing C++, and our clever CombineParticles framework, move to DaVinciTutorial4.

-- PatrickKoppenburg - 01 Oct 2007 -- PatrickKoppenburg - 11 Mar 2008 -- PatrickKoppenburg - 05 Jan 2009 -- JeremyDickens - 04-Jan-2011 -- PatrickSKoppenburg - 16-Oct-2012 -- PatrickSKoppenburg - 30-Sep-2013

--- This topic: LHCb > DaVinciTutorial1
Topic revision: r82 - 2013-09-30 - PatrickSKoppenburg