



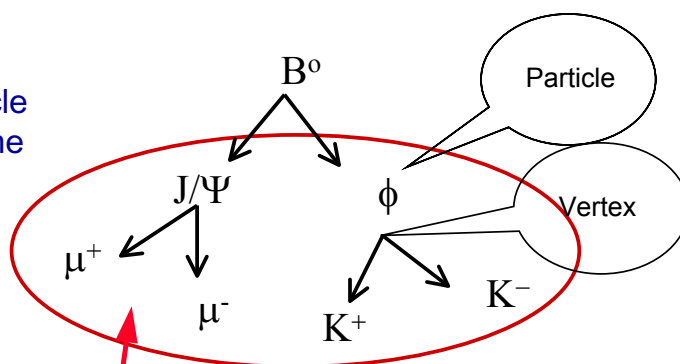
3

Beginning to Write a selection algorithm



A “Possible” Example

Make a 4 particle
vertex with some
constraints??



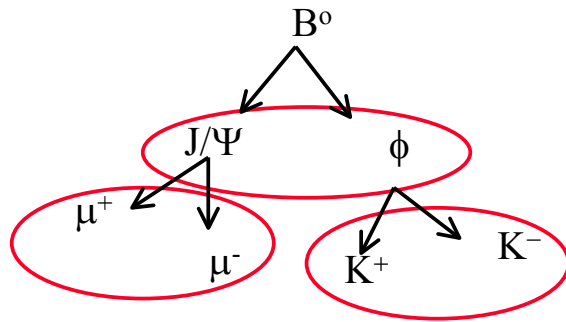
- **Two options:**
 - A selection algorithm that makes vertices of 4 particles with some cuts in the invariant mass of the lepton and Kaon Pairs
- **Three algorithms called in sequence**



A “Possible” Example (cont’d)

All are particle pairs with “roughly” the same algorithmic sequence

- Select two particles with some PiD
- Make a vertex
- Cut in some mass window and some “Geometrical variables”
- Use a switch for additional cuts in “decaying vertex point distance of daughters”
- Use the daughters for the B vertex



Can program a **Select2ParticleDecay** algorithm with carefully selected option



Two Main Components in a Selection

• Algorithms itself (the code)

- An “end user” will usually develop algorithms that will use DaVinci Tools

```
MySelAlgorithm::DVAlgorithm {
  declareProperty("CutName",cutVariable=defaultvalue);
}
MySelAlgorithm::initialize {
  Initialize all needed tools and services;
  Histograms;
}
MySelAlgorithm::execute {
  Where the work is done
}
MySelAlgorithm::finalize{
  Final statistics
}
```

Inherits from DVAlgorithm where the retrieving of the most frequently used tools is done

Recommended to make all analysis variable cuts available in job options.

Use emacs: it will create an Algorithm skeleton for you !

Naming convention: **SelectBd2Jpsi2mumu_Ks2pipi**
SelectJpsi2mumu_forBd2JpsiKs



Two Main Components in a Selection

- **Options file (its running configuration for a job)**
 - **Customize the tool used:**
 - PhysDesktop tool
 - the chosen ParticleMaker
 - the ParticleFilter and
 - each FilterCriterion
 - **Configure the user defined properties**
 - Cuts, histograms filling, etc
- **The same algorithm could be configured for different decays:**

```
ApplicationMgr.TopAlg+={Select2ParticleDecay/SelectJPsi};
ApplicationMgr.TopAlg+={Select2ParticleDecay/SelectPhi};
SelectXXXXX.MassWindowLoose = zzz;
SelectXXXXX.DaughterCandidates= {"Part1", "Part2"};
SelectXXXXX.MotherParticleName= {"Mother"};
.....
```

Naming convention:

Must specify the full decay chain

SelectBd2Jpsi2mumu_Ks2pipi



Select*.h

Since a Selection Algorithm inherits from the DVAlgorithm, it should include:

```
#include "DaVinciTools/DVAlgorithm.h"
```

and

```
class SelectJPsi2MuMu : public DVAlgorithm { ... }
```

The cuts will be properties of the algorithm, so they should be data members:

```
private:
```

```
double m_JPsiMassWin; ///< Mass window for the two Muons
```

```
double m_JPsiZWin; ///< Z vertex window for the two Muons
```

```
double m_chi2ConVtxCut; ///< chi2 of constrained vertex fit
```



Usually some histograms will be produced:

```
// Forward declarations
class IHistogram1D;
private:
bool m_produceHistogram; ///< flag for histo production
IHistogram1D* m_hSum4p; ///< Histo of two muons mass
IHistogram1D* m_hChi2ConFit; ///< Histo of the Chi2
```

some SdtHep information and counters:

```
long m_jpsiID; ///< SdtHep ID for JPsi
double m_jpsiMass; ///< SdtHep mass for JPsi
int m_nEvents; ///< N events processed
int m_JPsiCount; ///< Number of JPsi's
```



Select*.cpp

Files to be included:

```
// from Gaudi
#include "GaudiKernel/AlgFactory.h"
#include "GaudiKernel/SmartDataPtr.h"
#include "GaudiKernel/IDataProviderSvc.h"
#include "GaudiKernel/IParticlePropertySvc.h"
#include "GaudiKernel/ParticleProperty.h"
#include "GaudiKernel/IHistogramSvc.h"
#include "AIDA/IHistogram1D.h"
// from Event
#include "Event/EventHeader.h"
#include "Event/Vertex.h"
#include "Event/Particle.h"
```



```
// CLHEP
#include "CLHEP/Units/PhysicalConstants.h"
#include "CLHEP/Geometry/Point3D.h"
// local
#include "SelectJPsi2MuMu.h"
```



Constructor

Declare the Properties in the Constructor:

```
SelectJPsi2MuMu::SelectJPsi2MuMu( const std::string& name,
                                   ISvcLocator* pSvcLocator)
:   DVAlgorithm ( name , pSvcLocator),
    m_nEvents(0),
    m_JPsiCount(0){
declareProperty("HistogramFlag", m_produceHistogram = false);
declareProperty("JPsiMassWindow", m_JPsiMassWin = 0.2*GeV);
declareProperty("JPsiZWindow", m_JPsiZWin = 50.0 * cm);
declareProperty("Chi2ConFit", m_chi2ConVtxCut = 20.0);
}
```



initialize()

The `initialize()` method is where the tools are retrieved, the Particle Property Service is requested and the histograms are booked.

Remark: When a `StatusCode` is returned it is because it has the possibility of `FAILURE`, so it has always to be tested. I put here one example of testing, but it will be omitted from then on.

```
StatusCode SelectJPsi2MuMu::initialize() {
    MsgStream log(msgSvc(), name());
    log << MSG::DEBUG << "=="> Initialize" << endreq;
    StatusCode sc = StatusCode::SUCCESS;
```



```
// Load all necessary tools via the base class
sc = loadTools();
if( sc.isFailure() ) {
    log << MSG::ERROR << " Unable to load tools" << endreq;
    return StatusCode::FAILURE;
}
// Access the ParticlePropertySvc
IParticlePropertySvc* ppSvc = 0;
sc = service("ParticlePropertySvc", ppSvc);
ParticleProperty* partProp;
partProp = ppSvc->find( "J/psi(1S)" );
//Note that the particleID().pid() is the jetsetID() code
m_jpsiID = (*partProp).jetsetID();
m_jpsiMass = (*partProp).mass();
```



```

// If histograms are required initialize them
if( m_produceHistogram ) {
    m_hChi2ConFit = histoSvc()-> book("/stat/simple/5",
        "Chi2 of Constrained J/Psi vertex Fit",
        100, 0.0, 20.0);
    if( 0 == m_hChi2ConFit ) {
        log << MSG::ERROR << " Cannot register histogram 5"
            << endreq;
        return StatusCode::FAILURE;
    }
    m_hSum4p = histoSvc()->book("/stat/simple/6",
        "Mass of 2 Muons", 100, 3.05, 3.15);
}
return StatusCode::SUCCESS; }

```



finalize()

Print out some statistics

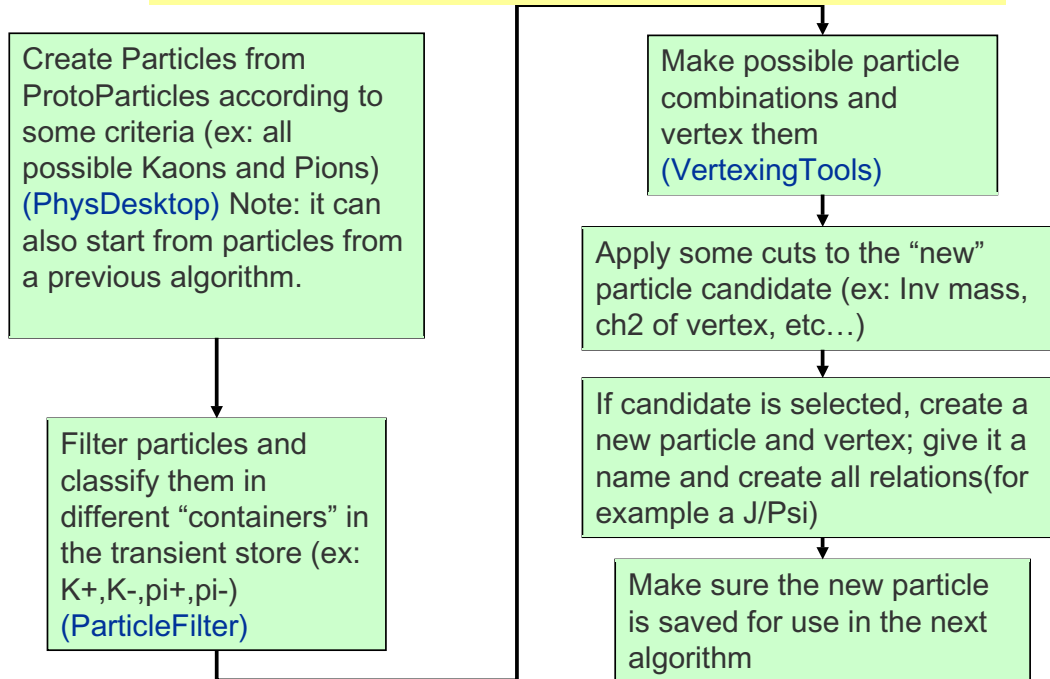
```

StatusCode SelectJPsi2MuMu::finalize() {
    MsgStream log(msgSvc(), name());
    log << MSG::DEBUG << "==> Finalize" << endreq;
    // Print out counters
    log << MSG::INFO << " Number of events processed = "
        << m_nEvents << endreq;
    log << MSG::INFO << " Number of selected JPsi = "
        << m_JPsiCount << endreq;
    // End of finalization step
    return StatusCode::SUCCESS;
}

```



The Execute Member Function



1.6

DaVinci Tutorial



Tools Needed

- **A creator of Particles from ProtoParticles and collector of existing Particles: PhysDesktop and ParticleMakers**
- **A Particle Filter Interface and several Filter Criteria implementations:**
 - PIDFilter
 - KinematicalFilter
 -
- **Vertexing algorithms, included constrained and unconstrained fit and the possibility to use the daughters in the vertexing.**
- **Tools to compute Geometrical variables**
 - Impact parameter
 - Distance between vertexes and particles.
 -
- **Particle transporter: Transports a particle to a given z**
- **Particle stuffer: create "consistent" particles from daughters**
- **Decay finders and Channel Selection Algorithms, Flavour Tagging**
- **MCDecay finder**
- **Debugging tool, Histograms, Ntuples**
- **Associators**
- **....**

1.7

DaVinci Tutorial



execute()

The execute() method is where event by event is processed.

```
StatusCode SelectJPsi2MuMu::execute() {
    MsgStream log( msgSvc(), name() );
    // Counter of events processed
    log << MSG::INFO << ">> Execute" << endreq;
    log << MSG::INFO << " processing event number " << ++m_nEvents
        << endreq;
    // Retrieve informations about event
    SmartDataPtr<EventHeader> evt(eventSvc(),
                                   EventHeaderLocation::Default );
    if ( evt ) {
        log << MSG::INFO << " retrieved EVENT: " << evt->evtNum()
            << " RUN: " << evt->runNum() << endreq;
```



```
}
else {
    log << MSG::ERROR << " not able to retrieve event" << endreq;
    return StatusCode::FAILURE;
}
//Fill the PhysDesktop particle and vertex vectors.
//Use the configuration set in the corresponding job options
StatusCode scDesktop = desktop()->getInput();
// Retrieve the particles and vertices from PhysDesktop
const VertexVector& verts = desktop()->vertices();
const ParticleVector& parts = desktop()->particles();
log << MSG::DEBUG << " Particle Vector size " << parts.size()
<< endreq;
// Print out some Primary Vertex Information
```



```
// Save the z position of the first one
VertexVector::const_iterator ivert = 0;
double zPrim = 0.;
for(ivert = verts.begin();ivert != verts.end();ivert++){
    if( (*ivert)->type() == Vertex::Primary ){
        log << MSG::DEBUG << "Primary Vertex z coordinate ="
            << " , " << (*ivert)->position().z() << endreq;
        HepSymMatrix primVertexErr = (*ivert)->positionErr();
        log << MSG::DEBUG <<"z error on prim vertex = "
            << sqrt(primVertexErr(3,3))/cm <<" cm" << endreq;
        zPrim = (*ivert)->position().z();
        break;
    }
}
```



```
// ParticleFilter according to job options
ParticleVector vMuPlus, vMuMinus;
StatusCode scFilPos = ParticleFilter()->
    filterPositive( parts, vMuPlus );
StatusCode scFilNeg = ParticleFilter()->
    filterNegative( parts, vMuMinus );
log << MSG::DEBUG<< "vMuPlus size" << vMuPlus.size()<<endreq;
log << MSG::DEBUG<< "vMuMinus size" << vMuMinus.size()<<endreq;
```



```

// Do all mu+/mu- combinations
ParticleVector::iterator iMuPlus;
ParticleVector::iterator iMuMinus;
for( iMuMinus = vMuMinus.begin();
      iMuMinus != vMuMinus.end(); iMuMinus++ ) {
  for (iMuPlus = vMuPlus.begin(); iMuPlus != vMuPlus.end();
        iMuPlus++) {
    // Find invariant mass
    HepLorentzVector twoMuComb(0.0, 0.0, 0.0, 0.0);
    twoMuComb = (*iMuMinus)->momentum() +
                (*iMuPlus)->momentum();
    //Units are MeV/mm/nsec. I want histos in GeV/cm
    if( m_produceHistogram )
      {m_hSum4p->fill( twoMuComb.m()/GeV, 1.);}

```



```

//Check that muon-antimuon invariant mass is close
to the J/Psi mass
if (fabs(twoMuComb.m() - m_jpsiMass)< m_JPsiMassWin){
  //Perform Unconstrained vertex fit
  Vertex MuMuVertex;
  StatusCode scMuMuVertex =
    vertexFitter()->fitVertex>(*iMuMinus),*iMuPlus),
                              MuMuVertex);
  log << MSG::DEBUG << "Unconstrained vertex position "
    << MuMuVertex.position().x()/cm << " "
    << MuMuVertex.position().y()/cm << " "
    << MuMuVertex.position().z()/cm << endreq;
  log << MSG::DEBUG << " Chisquare " << MuMuVertex.chi2()
    << endreq;

```



```
//Check that muon-antimuon vertex is within a
//reasonable window around z = 0
if ( fabs(MuMuVertex.position().z()) < m_JPsiZWin){
  // Cut on Chi2 of J/Psi unconstrained vertex fit
  if (MuMuVertex.chi2() < m_chi2UncVtxCut) {
    // Create Particle from Vertex (ParticleStuffer)
    Particle candJpsi;
    ParticleID jpsiPID( m_jpsiID );
    StatusCode scStuff =
      particleStuffer()->fillParticle( MuMuVertex,
        candJpsi, jpsiPID );
```



```
// Debug it, Access the daughters through the vertex
SmartRefVector<Particle>::const_iterator it;
for ( it = candJpsi.endVertex()->products().begin();
  it != candJpsi.endVertex()->products().end();
  it++ ){
  log << MSG::DEBUG << "Momentum of daughters "
    << (*it)->momentum().px() << " "
    << (*it)->momentum().py() << " "
    << (*it)->momentum().pz() << endreq;
}
```



```

//How to use the geometrical Displacement Tools.
double ip=0.;
Hep3Vector ipVector;
HepSymMatrix errMatrix;
double ipErr=0;
double dist=0.;
double distErr=0;
for(ivert = verts.begin();ivert != verts.end();ivert++){
  if ((*ivert)->type()==Vertex::Primary){
    // Calculate the IP vector(and its error) of the
    // mu- wrt primary vertex (other signatures are provided)
    StatusCode scImp = geomDispCalculator()->
      calcImpactPar>(*iMuMinus),>(*ivert),
      ip,ipErr,ipVector,errMatrix);

```



```

log« MSG::DEBUG« " IP = "« ipVector.mag()« endreq;
// Calculate the distance (and its error) of closest
// approach between two particles
StatusCode scCda = geomDispCalculator()->
  calcCloseAppr>(*iMuMinus),>(*iMuPlus)),
  dist, distErr);
log « MSG::DEBUG « " CDA = " « dist « endreq;
// Calculate the magnitude of the distance (and its
// error) between the primary and secondary vertex
StatusCode scDist = geomDispCalculator()->
  calcVertexDis(MuMuVertex,(*ivert),dist,distErr);
log « MSG::DEBUG « " DIST = " « dist « endreq;
}
}

```



```

//Perform mass constrained vertex fit
Vertex jpsiVtx;
Particle jpsi;
StatusCode scLagFit =
    massVertexFitter()->fitWithMass("J/psi(1S)",
        *(*iMuMinus),*(*iMuPlus), jpsiVtx, jpsi);
//Fill the Chi2 histogram
if( m_produceHistogram ) {
    m_hChi2ConFit->fill(jpsiVtx.chi2(), 1.);
}
//Cut on Chi2 of J/Psi constrained vertex fit
if (jpsiVtx.chi2() < m_chi2ConVtxCut) {
    log << MSG::INFO << " Passed all cuts " << endreq;
    m_JPsiCount++;

```



```

//saving THIS particle and its vertex to deskTop
//The pointer to the particle is returned ->
it is created
Particle* pInDesktop= desktop()->createParticle(&jpsi);
if( pInDesktop) {
    log << MSG::DEBUG << "J/Psi added to PhysDesktop "
}
else {
    log <<MSG::DEBUG<< "not able to save J/Psi in desktop"
        << endreq;
}
//Close all the if's
} //for(iMuMinus = vMuPlus.begin() ...
} //for( iMuPlus = vMuMinus.begin()...

```



```
// Now save desktop to TES in the location specified in
jobOptions
// Notice that this delete particles from desktop
  at the moment
// It can only be called once per Algorithm
scDesktop = desktop()->saveDesktop();
if (scDesktop) {
  log « MSG::INFO « " PhysDeskTop Saved to TES"«endreq;
}
else {
  log « MSG::ERROR « "not able to save desktop in TES"
  « endreq;
  return StatusCode::FAILURE;
}
```



```
// End of execution for each event
return StatusCode::SUCCESS;
}
```

