

---

---

## Particle Filter

**Filters an input vector of Particle objects, producing an output sub-vector of those Particles which pass a list of user defined criteria.**

☞ **The PhysDesktop is not modified!**

### Interface: IParticleFilter

```
StatusCode filter( const ParticleVector& input,
                  ParticleVector& output );
StatusCode filterPositive( const ParticleVector& input,
                           ParticleVector& output );
StatusCode filterNegative( const ParticleVector& input,
                           ParticleVector& output );
```



---

---

## Particle Filter (cont)

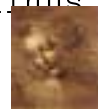
### Concrete Class: ParticleFilter

**One property - CriteriaNames - a vector of strings, where each string is a concrete filter criterion class name**

```
SelectJPsiMuMu.ParticleFilter.CriteriaNames =
    { "PIDFilterCriterion", "KinFilterCriterion" };
```

**Usage:(One ParticleFilter is provided in the DVAlgorithm:)**

```
const ParticleVector& parts = desktop()->particles();
ParticleVector vMuPlus, vMuMinus;
StatusCode scFilPos =
    particleFilter()->filterPositive( parts, vMuPlus);
StatusCode scFilNeg =
    particleFilter()->filterNegative( parts, vMuMinus );
```



---

## Particle Filter (cont)

**But various instances with different criteria can be requested and used in the same algorithm. Suppose you want to use separately the particle ID CL cut and the kinematical cuts: include in your Select\*.h:**

```
// Forward declarations
class IParticleFilter;
private:
IParticleFilter* m_pFilterMuons;
IParticleFilter* m_pFilterKin;
std::string m_FilterMuonsName;
std::string m_FilterKinName;
```



---

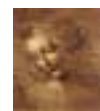
## Particle Filter (cont)

**in your Select\*.cpp:**

```
#include "DaVinciTools/IParticleFilter.h"
declareProperty( "ParticleFilter1", m_FilterMuonsName =
    "MuonFilter" );
declareProperty( "ParticleFilter2", m_FilterKinName =
    "MomentumFilter" );
```

**in the initialize() method:**

```
// Retrieve the ParticleFilter tool
sc = toolSvc()->retrieveTool("ParticleFilter",
    m_FilterMuonsName, m_pFilterMuons, this);
sc = toolSvc()->retrieveTool("ParticleFilter",
    m_FilterKinName, m_pFilterKin, this);
```



---

## Particle Filter (cont)

### and in the execute method:

```
ParticleVector vMuons;
//Fill the vMuons with all muons with CL > 5
StatusCode scFilMuons = m_pFilterMuons->filter(parts, vMuons);
ParticleVector vMuPlus, vMuMinus;
// Fill the vMuPlus with mu+ with pt > 1. GeV
StatusCode scFilPos = m_pFilterKin->filterPositive( vMuons,
    vMuPlus );
// Fill the vMuMinus with mu- with pt > 1. GeV
StatusCode scFilNeg = m_pFilterKin->filterNegative( vMuons,
    vMuMinus );
```



---

## Particle Filter (cont)

### with the following configuration:

```
SelectJPsiMuMu.ParticleFilter1 = "MuonFilter";
SelectJPsiMuMu.ParticleFilter2 = "MomentumFilter";
SelectJPsiMuMu.MuonFilter.CriteriaNames =
    { "PIDFilterCriterion" };
SelectJPsiMuMu.MomentumFilter.CriteriaNames =
    { "KinFilterCriterion" };
SelectJPsiMuMu.MuonFilter.PIDFilterCriterion.ParticleNames
    = { "mu-", "mu+" };
SelectJPsiMuMu.MuonFilter.PIDFilterCriterion.ConfidenceLevels
    = { 0.05, 0.05 };
SelectJPsiMuMu.MomentumFilter.KinFilterCriterion.MinPt = 1000;
```



---

## FilterCriterion

**Tests whether a Particle satisfies a certain criterion.  
There may be any number of different filter criterion classes.  
Each one implements directly the IFilterCriterion interface:**

```
bool isSatisfied( const Particle* const& );  
bool operator()( const Particle* const& );
```

**Two concrete filter criterion are provided:**

**PIDFilterCriterion: selects Particles with a given ID and CL  
Configuration:**

```
SelectJPsiMuMu.ParticleFilter.PIDFilterCriterion.  
    ParticleNames = { "mu-", "mu+" };  
SelectJPsiMuMu.ParticleFilter.PIDFilterCriterion.  
    ConfidenceLevels = { 0.05,0.05 };
```



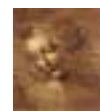
---

## FilterCriterion (cont)

**KinFilterCriterion: selects Particles with a minimum momentum and a minimum transverse momentum.**

**Configuration:**

```
SelectJPsiMuMu.ParticleFilter.KinFilterCriterion.MinMomentum  
= 1.0;  
SelectJPsiMuMu.ParticleFilter.KinFilterCriterion.MinPt = 1.0;
```



---

## How to write a FilterCriterion Tool

☞ Use emacs: it makes the tool template for you.

☞ Implements the IFilterCriterion Interface

**Example of KinFilterCriterion.h:**

```
#include "GaudiKernel/AlgTool.h"
#include "DaVinciTools/IFilterCriterion.h"
class KinFilterCriterion : public AlgTool,
                          virtual public IFilterCriterion {
public:
  /// Standard constructor
  KinFilterCriterion( const std::string& type,
                    const std::string& name,
                    const IInterface* parent);
  /// Destructor virtual KinFilterCriterion( ){};
  /// Test if kinematical filter (minimum momentum and pt) is satisfied.
  inline bool isSatisfied( const Particle* const & part );
```



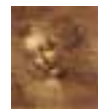
---

```
/// Test if kinematical filter (minimum momentum and pt) is satisfied.
inline bool operator()( const Particle* const & part );
```

```
private:
double m_minMom; ///< Minimum momentum
double m_minPt; ///< Minimum pt
};
```

**KinFilterCriterion.cpp**

```
// from Gaudi
#include "GaudiKernel/ToolFactory.h"
#include "GaudiKernel/MsgStream.h"
// local
#include "KinFilterCriterion.h"
//-----
// Implementation file for class : KinFilterCriterion
//
// 19/03/2002 : Paul Colrain
//-----
```



---

```

// Declaration of the Tool Factory
static const ToolFactory<KinFilterCriterion> s_factory ;
const IToolFactory& KinFilterCriterionFactory = s_factory ;
//=====
// Standard constructor, initializes variables
//=====
KinFilterCriterion::KinFilterCriterion( const std::string& type,
                                       const std::string& name,
                                       const IInterface* parent )
: AlgTool ( type, name , parent ) {
// declare additional interface
declareInterface<IFilterCriterion>(this);
// declare properties
declareProperty( "MinMomentum", m_minMom = 0. );
declareProperty( "MinPt", m_minPt = 0. );
}
//

```




---

```

//=====
// Test if filter is satisfied
//=====
bool KinFilterCriterion::isSatisfied( const Particle* const & part )
{
return ( part->momentum().vect().mag() > m_minMom &&
        part->momentum().vect().perp() > m_minPt );
}
//=====
// Test if filter is satisfied
//=====
bool KinFilterCriterion::operator()( const Particle* const & part ) {
return ( part->momentum().vect().mag() > m_minMom &&
        part->momentum().vect().perp() > m_minPt );
}
//=====

```

