LCFIVertexPackage

Generated by Doxygen 1.8.6

Fri Dec 2 2016 12:36:42

Contents

1 The LCFI Vertex Package

The LCFI Vertex Package provides the vertex finder ZVTOP, originally developed for SLD by D. Jackson [1], flavour tagging as well as vertex charge determination for b- and c-jets. By default, the flavour tag provided is obtained from the algorithm by R. Hawkings [2]. It is based on a neural net approach, combining track and vertex information to distinguish b, c- and light jets. The algorithm to determine vertex charge follows the SLD-approach [3], with modifications for b-jets developed by S. Hillert [4].

In addition to the algorithms, the package provides an object-oriented framework, in which the default approach can easily be modified and extended. Care was taken to make all main parameters of the code accessible to the user as steering parameters. The code was interfaced to the MarlinReco analysis framework and uses LCIO for input and output, permitting it to be used in conjunction with algorithms from other reconstruction frameworks.

The code was implemented by Ben Jeffery (ZVTOP, LCIO/Marlin interface, working classes design and testing), Erik Devetak (Flavour tag inputs calculation and testing, MC Jet flavour, Vertex Charge Processor), Mark Grimes (Flavour tag procedure, Vertex fitter), Dave Bailey (neural network code), Victoria Martin (AIDA Plot Processor), Tomas Lastovicka (Kalman filter for vertex fitting), Kristian Harder (use GEAR-geometry for suppression of hadronic interactions, Purity - Efficiency macro) and Sonja Hillert (coordination, system test). The authors thank the LCFI physics group for help and advice during the development phase, in particular D. Jackson (advice on ZVTOP), K. Harder(testing, Mokka/Gear interface) V. Martin (test of vertex charge procedure for c-jets), T. Lastovicka (testing), R. Walsh (testing), Clare Lynch (testing).

We would also like to thank F. Gaede, T. Behnke and N. Graf for fruitful discussions, D. Martsch for producing a test sample on the GRID and A. Raspereza for advice and for extending the track cheater functionality to provide the input required by the Vertex Package.

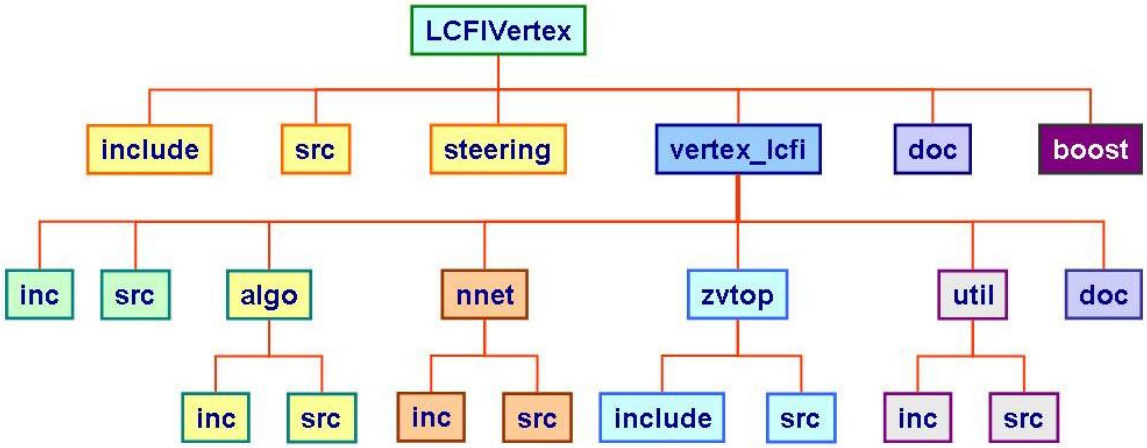

Figure 1: Vertex Package directory structure

Directories of the package are organised as follows: the top-level directories contain the Marlin processors (src) and the include- and steering files they require, as well as macros (macro) that can be run on output created by some of the processors. For example, a root macro is provided to make comparison plots of purity vs efficiency obtained from two subsequent runs of the PlotProcessor. (Note, that the package provides root output only if compiled with root option). The top-level directories provide an interface to the main part of the code, which is located in the directory vertex_lcfi. The Marlin processors access a set of algorithm classes for ZVTOP, flavour tag and vertex charge calculation which can be found in the subdirectory algo. These algorithm classes all inherit from a simple interface Algo, providing parameters and the method "calculateFor", returning the output of the algorithm. Input to the algorithm classes are objects like jets or events. The implementation of these object classes can be found in the directories vertex_lcfi/inc and vertex_lcfi/src. Working classes specific to the vertex finder ZVTOP, providing functionality like vertex finding, vertex resolving and vertex fitting, are located in the directory vertex_lcfi/zvtop. The neural network code is kept in the directory vertex_lcfi/nnet.

The following Marlin processors are provided:

- TrueAngularJetFlavourProcessor: provides the true jet flavour using MC information
- PerEventIPFitterProcessor: determines the event vertex (IP)
- RPCutProcessor: flexible processor for applying various track selection cuts
- ZVTOPZVRESProcessor: find vertices running the ZVRES branch of ZVTOP
- ZVTOPZVKINProcessor: find vertices running the ZVKIN branch of ZVTOP (ghost track algorithm)
- FlavourTagInputsProcessor: calculate input variables for the flavour tag neural net and the vertex charge
- NeuralNetTrainerProcessor: train neural networks for flavour tag
- FlavourTagProcessor: use pretrained neural nets to obtain flavour tag
- VertexChargeProcessor: calculate vertex charge for b- and/or c-jets
- PlotProcessor: calculate purity and efficiency and produce performance plot (if compiled with root)
- LCFIAIDAPlotProcessor: diagnostic plots and tables for flavour tag inputs and outputs

The example steering files provided show how the package could be run in a typical analysis. The order in which the steering files would be called is as follows:

- `cheattracks+jetfind.xml` - note that hit collection names in this file are geometry specific, the default detector geometry assumed in this example file is LDC01_05Sc.

- `truejetflavour.xml`
- `ipfit.xml`
- `zvres.xml`
- `fti.xml`
- `trainNeuralNets.xml` (optional, only needed for special training run to obain new flavour tag neural nets)
- `ft.xml`
- `Bvertexcharge.xml`
- `Cvertexcharge.xml`
- `ftplot.xml`

The first of these steering files calls event reconstruction processors from MarlinReco that are not part of the package but need to be run in order to obtain the collections required. Running Marlin with this steering file creates an input LCIO file for the package, by default called cheatout.slcio. It contains collections with MC particles, jets and the ReconstructedParticles within the jets. The execution flow diagram shows how these collections are used by the LCFIVertex package, as well as processors used and collections created if using the example steering files above. (The training of new neural nets is not covered in the diagram, as this would be done in a dedicated training run; the typical application uses networks that have already been trained). Unless otherwise indicated, all collections shown in the diagram are of type ReconstructedParticle.

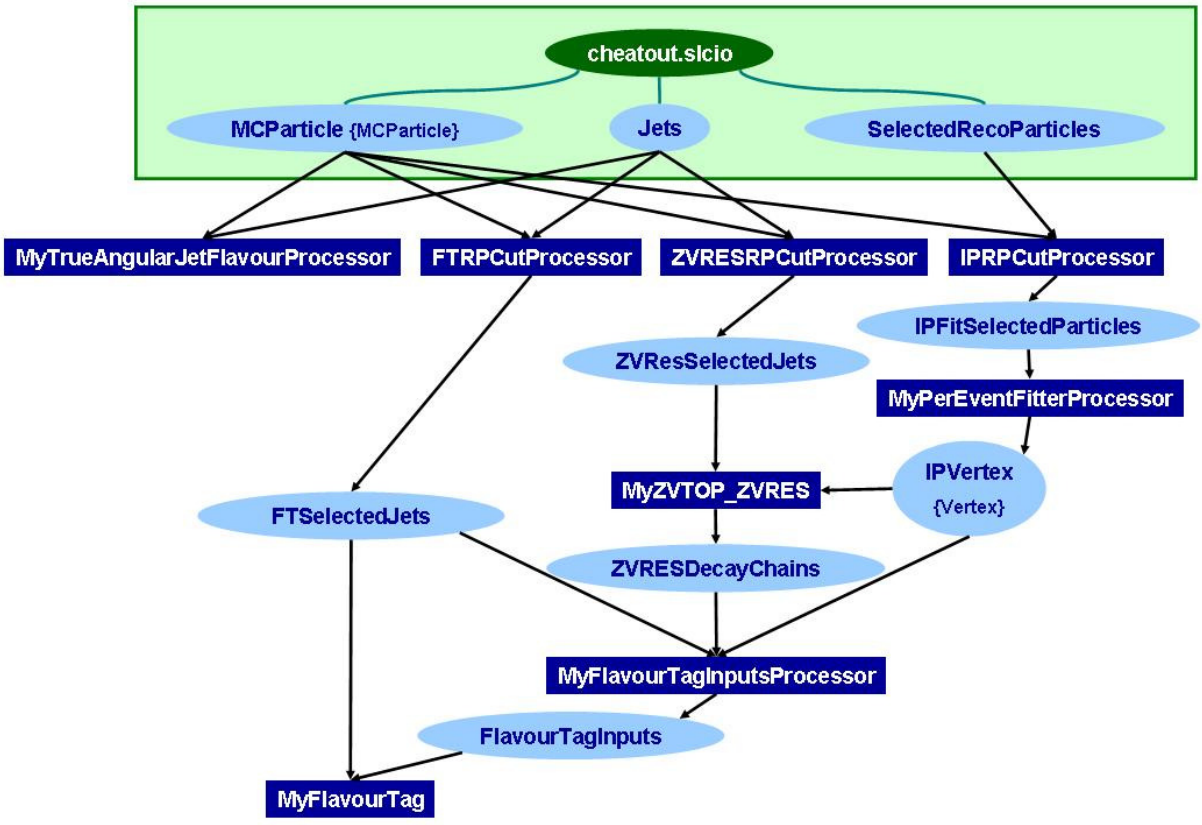

Figure 2: Vertex Package execution flow obtained from the example steering files

Some processors depend on others to have run before, e.g. ZVTOP and the flavour tag inputs processor each require their own track selection implemented by the RPCutProcessor. Further details are provided in the documentation for each of the processors (see above) and in the tutorial section. Information on the internal ZVTOP

classes can be found in the ZVTOP documentation. Scope and usage of the neural network code (which can also be used for purposes other than flavour tagging) is described in the neural net documentation.

In addition to code and example steering files, the package provides a set of pre-trained networks for the Hawkings default flavour tag. These have been trained using the fast MC SGV and are located in a new repository **tagnet** in the Zeuthen CVS repository (Use 'tagnet' as project name when checking out the directory). **We strongly recommend submitting any new networks that users train with different boundary conditions to this directory along with a detailed description of training conditions.** A form for providing training information can be found in the same directory in order to make ILC physics studies more transparent and ease comparisons of analyses performed within different groups, frameworks or detector concepts.

Summary of changes in release versions:

- v00-02-01

[1] D. Jackson, NIM A 388 (1997) 247

[2] R. Hawkings, LC-PHSM-2000-021

[3] J. Thom, SLAC-R-585 (2002), T. Wright, SLAC-R-602 (2002)

[4] S. Hillert, proceedings LCWS 2005

In case of comments or questions **not answered by the documentation** please contact the development and maintenance team:

Erik Devetak (E.Devetak1@physics.ox.ac.uk)

Mark Grimes (Mark.Grimes@bristol.ac.uk)

Kristian Harder (K.Harder@rl.ac.uk)

Sonja Hillert (S.Hillert1@physics.ox.ac.uk)

Talini Pinto Jayawardena (T.S.Pinto.Jayawardena@rl.ac.uk)

Ben Jeffery (B.Jeffery1@physics.ox.ac.uk)

Tomas Lastovicka (T.Lastovicka1@physics.ox.ac.uk)

Clare Lynch (Clare.Lynch@bristol.ac.uk)

Victoria Martin (victoria.martin@ed.ac.uk)

Roberval Walsh (r.walsh@ed.ac.uk)

2 LCIO Interface

Description of this package's use of LCIO to store results of processors.

2.1 Storage of Vertexing Result

The processors ZVTOPZVKINProcessor and ZVTOPZVRESProcessor

both store their results in the same way. They provide information on the decay vertices found, stored in a collection of LCIO Vertices and a collection of ReconstructedParticles, representing decaying particles that give rise to these vertices, as described in Frank Gaede's Frank Gaede's forum post

In LCIO, Vertices and ReconstructedParticles, are connected as follows:

- Each decay vertex found has a corresponding LCIO::Vertex.
- Each decay vertex found also has a corresponding LCIOReconstructedParticle which respresents the decaying particle and holds kinematic information.
- This accompanying decaying ReconstructedParticle is accessed though Vertex::getAssociatedParticle()

- The descendant tracks which are produced by the particle are attached to the decaying ReconstructedParticle and accessed through ReconstructedParticle::getParticles()
- Each ReconstructedParticle points to its start and end vertex (if any) through ReconstructedParticle::getStartVertex() and ReconstructedParticle::getEndVertex()

In essence we end up with three types of objects with links between them; Vertices, Decaying ReconstructedParticles, Stable ReconstructedParticles.

Note that the information stored in the ReconstructedParticles created using ZVTOP information differs from the one of those created by the track reconstruction code. The getStartVertex and getEndVertex methods permit building up a representation of a heavy flavour decay chain. In the current ZVTOP version, this is reconstructed as follows: vertices are sorted by increasing radius from the IP, the start vertex of the accompanying ReconstructedParticle is set to point to the previous vertex in the collection in this order. Note that this is only an approximation of the decay chain in an actual physics event, which may differ or be more complex (e.g. one decay vertex may give rise to two instable particles. If correctly reconstructed, their ReconstructedParticles would point to the same start vertex. Of the corresponding ZVTOP vertices, only the one closer to the IP will point to its correct start vertex, while the further one will point to the nearer one instead.

As each RP points to its vertices, to store more than one vertexing result it is necessary to take a copy of the set of RPs that are in each vertex. Each copy points to the original unique RP through ReconstructedParticle::getParticles()[0].

A master ReconstructedParticle object is created that points to all decaying and stable ReconstructedParticles in the decay chain though ReconstructedParticle::getParticles(). The ReconstructedParticle::getStartVertex() is set to the first Vertex in the decay chain (usually the IP). This is the main object for accessing the decay chains as it allows one to iterate over all the ReconstructedParticles contained within.

These objects are then stored in three collections:

- DecayChainRPTracksCollectionName: default name: ZV***DecayChainRPTracks, stores decaying RPs and copies of input RPs for all decay chains.
- VertexCollection: default name: ZV***Vertices, stores the Vertices for all decay chains.
- DecayChainCollectionName: default name: ZV***DecayChains, stores the master ReconstructedParticle DecayChainCollectionName, in the same order as the Jets that were input to the algoithm in JetRPCollection

Example

If you wanted to print out the d0 of each LCIO::Track in each Vertex of a decay chain found for the second jet in the jet collection:

```
//Get the decay chain master RP collection and get the second decay chain
LCCollection* DecayChainRPCol = evt->getCollection( _DecayChainRPColName );
ReconstructedParticle* DecayChainRP = dynamic_cast<ReconstructedParticle*>(DecayChainRPCol->getElementAt(1)
//Make a list of vertices
vector<lcio::Vertex*> LCIOVertices;
//Add the primary first
LCIOVertices.push_back(DecayChainRP->getStartVertex());
//Loop over RPs to find all the other vertices
vector<ReconstructedParticle*> RPs = DecayChainRP->getParticles();
for (vector<ReconstructedParticle*>::const_iterator iRP = RPs.begin();iRP < RPs.end();++iRP)
{
         lcio::Vertex* MyVertex = (*iRP)->getStartVertex();
         if(MyVertex)
         {
                  vector<lcio::Vertex*>::const_iterator it = find(LCIOVertices.begin(),LCIOVertices.end(),
      MyVertex);
                  if (it == LCIOVertices.end())
                  {
                           LCIOVertices.push_back(MyVertex);
                  }
         }
}
//Loop over the vertices
for (size_t i = 0; i < LCIOVertices.size(); ++i)
{
        std::cout << "Vertex " << i << "has d0's:" << std::endl;
```

```
        //Get the Vertex RPs from the Vertices decaying RP
        std::vector<ReconstuctedParticle*> VertexRPs = LCIOVertices[i]->getAssociatedParticle()->
      getParticles();
        for (size_t j = 0; j < VertexRPs.size(); ++i)
        {
                //Get the Track (remember the Vertex RP is a copy which points to the original)
                std::cout << VertexRPs[j]->getParticles()[0]->getTracks()[0]->getD0() << ",";
        }
        std::cout << std::endl;
}
```

Storage of track chi squareds

If OutputTrackChi2 is set to true the vertexing processors will output the chi squared each track contributes to its vertex.

This is stored in a collection named: TrackRPCollectionName+"TrackChiSquareds" ie. The name of the first collection appended with "TrackChiSquareds"

The chi squareds are stored as a collection of LCFloatVecs in the same order as the tracks in DecayChainRPTracksCollection. Currently the LCFloatVec contains one value - the chi squared in the start vertex, but the end vertex could be supported if needed as a second value.

2.2 Storage of Flavour Tag Inputs and Flavour Tag Result

The lists of variables produced by the Flavour Tag and Inputs are stored in collections of LCFloatVecs, with one LCFloatVec per jet. Within the LCFloatVec, the names and order of the variables are stored in the parameter of the run header as a string vector stored under the name of the LCFloatVec collection. Note that the same LCFloatVec is used for the flavour tag inputs, the vertex charge and further additional variables.

Example

Get the secondary vertex probability for the second jet:

```
//Get the variable names in the FlavourTagInputsCollection
std::vector<std::string> VarNames;
(pRun->parameters()).getStringVals(_FlavourTagInputsCollectionName,VarNames);
//Convert this to a convenient map
std::map<std::string,unsigned int> IndexOf;
for (size_t i = 0;i < VarNames.size();++i)
{
        IndexOf[VarNames[i]] = i;
}
//Get the inputs for the second jet
lcio::LCCollection* pInputs=pEvent->getCollection( _FlavourTagInputsCollectionName );
LCFloatVec* FTInputs = dynamic_cast<lcio::LCFloatVec*>( pInputs->getElementAt(1) );
</pre>
Get the Secondary Probability by indexing the LCFloatVec
<pre>
double SecProb = (*FTInputs)[IndexOf["SecondaryVertexProbability"]];
```

Ben Jeffery - b.jeffery1@physics.ox.ac.uk

3 Description of Track Selection Cuts

The track selection cuts are mainly from LC note LC-PHSM-2000-021 with minor changes.

These have not yet been optimised for full reconstruction.

3.1 IP Fitting Cuts

Details to follow - for now see steering file ipfit.xml

3.2 ZVTOP Cuts

Details to follow - for now see steering file zvres.xml

3.3 FlavourTagInputs Cuts

Details to follow - for now see steering file fti.xml

4 Neural Net Package

4.1 Acknowledgements

All code in this neural net package was written by David Bailey of the University of Manchester.

4.2 Assumed knowledge

- Standard Template Library (STL) vectors.

4.3 Remarks

- All neural net classes are in the namespace `nnet`.
- This is by no means a complete guide to every feature available, at present at least.

4.4 Basic principles of an artificial neural network

This is a very basic introduction to the principles of a neural network (geared specifically at the way this package works). If you have any experience with neural networks you can safely skip this section.

Neurons are created to accept an arbitrary number of inputs, and based on these provide a single output value. The output is given by the neurons $thresholdfunction$, which can be any given function of the neurons $activationvalue$ (see the Neuron Descriptions for the functions actually provided with this package).

The activation value is given by multiplying each input by a pre calculated $weight$ depending on how important that input is, and summing these results. Each neuron can also be given a bias, depending on how important that neuron is to the network, but more on that later.

Calculating these weights is the important part, and is what differentiates a well performing network from a bad one. This process is known as $training$, and is performed by a training algorithm (see Training the network for the algorithms provided here). Basically, you provide the training algorithm with a set of data that you know the answers to (the result you would want the network to give you), and it changes the weights to give the best possible results for all the elements in the data set.

As a basic example, imagine a network composed of a single neuron that tells you if a food is bad for you or not. Say it is set up with three inputs, fibre content, fat content and colour. For simplicity, lets give the neuron a linear threshold function, so just a function that multiplies the activation value by a set constant, say k. The output of the network would be

$$output = f(activationvalue) = k \times (fibrecontent \times weight_{fibre} + fatcontent \times weight_{fat} + colour \times weight_{colour})$$

The network is useless until the values of the weights are adjusted so that they give an accurate output. To do this, a large database of foods is required where the properties of colour, fibre and fat content are known, as well as some reliable value as to how healthy the food is. The training algorithm then modifies the weights to try and get the best match of the output to the expected value for each food in the database. When somebody comes along with a new food, its properties can be put into the network and a (hopefully) reliable value as to how healthy it is pops out the other end.

Ideally, once trained, the weight given to colour will be zero since that is completely irrelevant (ignoring artificial colourings). However, if the training sample has just a few blue foods, which just happen to be bad for you, then the training algorithm will wrongly ascribe a high weight.to the colour input. Also, if the training sample foods have pretty similar fat and fibre contents, but are radically differently healthy (say, maybe due to salt content), then the training algorithm will probably be unable to make any sense of the sample, and give useless weights. This emphasises the need to select a large and varied training sample (as well as setting up the network with meaningful inputs in the first place).

Realistically, a network will be composed of many neurons so that all 'cross effects' between the inputs are taken into account (where a weighting for one input needs to depend on other inputs as well). Here, the network would be built up with layers of neurons where the input for each neuron in a layer is the output from each neuron in the layer before. The final layer would have just one neuron, so that you get just one output for the network.

4.5 Creating and training a new neural net

The method used to create a new network varies slightly depending on the algorithm used to train it. Sections Building the neuron layers and Creating the network describe how to setup the network ready for training, which is common to all training algorithms. The `BatchBackPropagationAlgorithm`, `BackPropagationCGAlgorithm` and `GeneticAlgorithm` algorithms require the training data to be pre-stored in a `nnet::NeuralNetDataSet` class (section Building the training sample), and will train themselves over the whole data set. `BackPropagationAlgorithm` on the other hand performs one training step at a time to provide more control over each training step.

Descriptions of the algorithms are given in Training the network.

4.5.1 Building the neuron layers

Only simple nets can be built, where each neuron takes the outputs of all of the neurons in the previous layer as its inputs. Details about the neurons behaviour are given in Neuron Descriptions.

There are two methods, one where neurons can have different types, and a simpler one where all of the neurons have the same type.

4.5.1.1 All neurons of the same type

Building the neuron layers simply consists of creating an STL vector of integers with the number of neurons in each layer, including the output layer but excluding the input layer. The type of all of the neurons is set later when the network is built. So if a network takes 3 inputs, has two hidden layers with 6 neurons and 4 neurons respectively, and 2 outputs the layers would be set like this:

```
std::vector<int> neuronsInLayer;
neuronsInLayer.push_back( 6 );
neuronsInLayer.push_back( 4 );
neuronsInLayer.push_back( 2 );
// The number of inputs is set later.
```

4.5.1.2 Neurons with different types}

These are set in a similar way, but instead of integers specifying the number of neurons in each layer, another STL vector of strings specifying the name of each neuron type is used, with the number of neurons set by the size of the vector. Currently available types (descriptions are given in Neuron Descriptions) are:

```
LinearNeuron
SigmoidNeuron
TanSigmoidNeuron
```

So if a network as in the previous example is to be built (with arbitrary neuron types):

```
std::vector<std::string> layer1;
layer1.push_back( "LinearNeuron" );
layer1.push_back( "SigmoidNeuron" );
```

```
// ...and so on until there are six names in the vector -> six neurons in the layer

std::vector<std::string> layer2;
layer2.push_back( "TanSigmoidNeuron" );
// ... and so on another three times

std::vector<std::string> outputlayer;
outputlayer.push_back( "LinearNeuron" );
outputlayer.push_back( "LinearNeuron" );

std::vector< std::vector<std::string> > neuronNames;
neuronNames.push_back( layer1 );
neuronNames.push_back( layer2 );
neuronNames.push_back( outputlayer );
// The number of inputs is set later.
```

4.5.2 Creating the network

Once the layer structure has been set up, the network can be created as follows, depending on which layer specification method was used.

4.5.2.1 All neurons of the same type

The type of the neurons is set by creating a neuron builder and passing its address to the network constructor. The names of available builders are the same as for the neurons, but with <tt>Builder</tt>" on the end, for examplennet::LinearNeuronBuilder" will build "LinearNeuron"s.

```
int numberOfInputs=3; // number of inputs set when the network is created
nnet::SigmoidNeuronBuilder myNeuronBuilder; // the type of ALL the neurons is set here

// now create the network using the neuronsInLayer vector from before
nnet::NeuralNet sameNeuronsNet( numberOfInputs, neuronsInLayer, &myNeuronBuilder );
```

4.5.2.2 Neurons with different types}

All that is needed here is the STL vectors of neuron names previously initialised and the number of inputs.

```
int numberOfInputs=3; // number of inputs set when the network is created

// now create the network using the neuronNames vector from before
nnet::NeuralNet differentNeuronsNet( numberOfInputs, neuronNames );
```

4.5.2.3 Random number generation

In each case, you can optionally use a random seed for the random number generator that sets the initial neuron weights by adding a boolean parameter at the end of the constructor arguments. Default is to use a random seed.

```
//don't use a random seed
nnet::NeuralNet sameNeuronsNet( numberOfInputs, neuronsInLayer, &myNeuronBuilder, false );

//use a random seed (default)
nnet::NeuralNet differentNeuronsNet( numberOfInputs, neuronNames, true );
```

Note that currently **the default implementation uses `rand()` for random numbers**, and the random seed is taken from the current system time. If you require something more sophisticated modify the "RandomNumberUtils.h" file.

4.5.3 Building the training sample

A network can be trained without setting out the data sample into a nnet::NeuralNetDataSet using the BackPropagationAlgorithm, but large scale training is easiest using the other algorithms so this will be covered here.

Data is added to the nnet::NeuralNetDataSet by calls to addDataItem, with a vector of inputs and a vector of the expected outputs as the arguments. All items in the data set must have the same number of inputs and

outputs; the first item you add sets these sizes for the whole data set. If you try and add an item where the input or output vectors are not the correct size, then an error will be printed to standard error and the item will be ignored.

For example:

```
// for a network to calculate the probability a given animal is a donkey
// with inputs, in order, of "number of legs", "height" and "length of tail"
nnet::NeuralNetDataSet \label{donkeyNetDataSet}animalSample;
std::vector<double> inputs;
std::vector<double> output;

// for donkey 1
output.push_back( 1 ); // I know for certain this animal is a donkey
inputs.push_back( 4 ); // it has four legs
inputs.push_back( 1.45 ); // it is 1.45m tall
inputs.push_back( 0.32 ); // it has a tail 32cm long

// This call sets the sample to demand 3 inputs and 1 output
animalSample.addDataItem( inputs, output ); // Add this animal to the training sample

inputs.clear(); // Clear all the data so that the vectors can be reused
output.clear();

//for Geoff
output.push_back( 0 ); // I know for certain Geoff isn't a donkey
inputs.push_back( 2 ); // he has 2 legs
inputs.push_back( 1.82 ); // he is 1.82m tall

// This will not be added, because there are not enough inputs!
animalSample.addDataItem( inputs, output ); // Add Geoff to the training sample

// To add Geoff to the training sample we need to match the number of inputs
inputs.push_back( 0 ); // Geoff's tail is 0cm long
animalSample.addDataItem( inputs, output ); // This will now work
```

4.5.4 Training the network

To train the network, a training algorithm is created with the network to be trained as the constructor argument, and a call to train is made with the number of training epochs and the training data. Currently available training algorithms are:

```
nnet::BackPropagationAlgorithm
nnet::BackPropagationCGAlgorithm
nnet::BatchBackPropagationAlgorithm
nnet::GeneticAlgorithm
```

4.5.4.1 Training with BackPropagationAlgorithm

The Back Propagation Algorithm uses the back propagation method for determining the gradient of the error, and then gradient descent to modify the weights to minimise the error. It is very similar to the BatchBackPropagation-Algorithm except that it only performs one training step at a time to give more control over the training parameters at each step.

The algorithm class is constructed by giving it the network to be trained, and optionally values for $learningRate$ and $momentumConstant$ (defaults are 0.5 for both). The $learningRate$ parameter is just a multiplier applied to the calculated change required for each weight, larger values will mean the weights will change more rapidly with each step. The previous steps' calculated change is also added to the current steps', but multiplied by the $momentumConstant$ value. A value greater than or equal to one for this would stop the algorithm settling on a maximum because (at least) the full previous change is added as well.

The `train` method is used to perform one training run, and returns the error. It takes a vector of the inputs and a vector of the required outputs, so if the first data item in the previous example is used for the step:

```
std::vector<double> inputs;
std::vector<double> output;

// for donkey 1
output.push_back( 1 ); // I know for certain this animal is a donkey
inputs.push_back( 4 ); // it has four legs
inputs.push_back( 1.45 ); // it is 1.45m tall
inputs.push_back( 0.32 ); // it has a tail 32cm long

//set learningRate to 0.6 and momentumConstant to 0.4
```

```
nnet::BackPropagationAlgorithm myTrainer( myPreviouslyCreatedNetwork, 0.6, 0.4 );

double errorForThisStep=myTrainer.train( inputs, output );
```

4.5.4.2 Training with BatchBackPropagationAlgorithm

This is essentially the same as `BackPropagationAlgorithm`, except it is supplied with a training sample which it will loop over itself. It can also be set do so repeatedly by specifying the number of epochs to run when calling the `train` method. The error for the most recent epoch is returned by `train`, and the errors from previous epochs can be retrieved as a vector with the `getTrainingErrorValuesPerEpoch` method.

```
//created in the same way as for the single step version
nnet::BatchBackPropagationAlgorithm myTrainer( myPreviouslyCreatedNetwork ) //use default learningRate and
       momentumConstant

//train using the sample in the previous example and 50 epochs
double finalError=myTrainer.train( 50, animalSample );

//get the errors from previous epochs to see how things are converging
std::vector<double> errors=myTrainer.getTrainingErrorValuesPerEpoch();
```

4.5.4.3 Training with BackPropagationCGAlgorithm

This algorithm is similar to `BatchBackPropagationAlgorithm` except that it uses the conjugate gradient method to minimise the error instead of gradient descent. It offers three types of function to calculate the β coefficient (see any detailed description of conjugate gradients) selected using the `setBetaFunction` method. These are `FletcherReves"`, PolakRibiere", and "ConjugateGradient", used as an enumeration as quoted. The default is FletcherReves.

```
//created in the same way as for the single step version
nnet::BackPropagationCGAlgorithm myTrainer( myPreviouslyCreatedNetwork )

//set the beta function to Polak-Ribiere
myTrainer.setBetaFunction( nnet::BackPropagationCGAlgorithm::PolakRibiere );

//train using the sample in the previous example and 500 epochs
double finalError=myTrainer.train( 500, animalSample );
```

4.6 Obtaining results

To get results from the neural network, the output method takes the inputs as an STL vector of doubles, and provides the results as an STL vector of doubles. So to determine if some animal is a donkey using a network trained from data of form of the data set in the previous example:

```
// Using the previously trained net "donkeyNet" to
// find out if Fido is a donkey...
std::vector<double> inputs;
inputs.push_back( 4 ); // Fido has four legs
inputs.push_back( 0.54 ); // he is 54cm tall
inputs.push_back( 0.25 ); // his tail is 25cm long

std::vector<double> output=donkeyNet( inputs );
// The network was set to have only one output, if there were any
// more then they would be the higher elements in the vector.
std::cout << "Likelihood Fido is a donkey= " << output[0] << std::endl;
```

4.7 Saving a neural net to disk

Neural nets can either be saved as plain text or XML files, with the default being XML. To choose between the two make a call to `NeuralNet::setSerialisationMode` with either `nnet::NeuralNet::PlainText` or `nnet::NeuralNet::XML`.

The network can then be saved to disk by passing a C++ stream to serialise. For example:

```
myNeuralNet.setSerialisationMode( nnet::NeuralNet::XML );
std::ofstream outputFile( "/home/me/myNeuralNet.xml" );
myNeuralNet.serialise( outputFile );
```

The network can also of course be printed to standard output by calling `serialise( std::cout )`.

4.8 Loading a neural net from disk

A network can be loaded from disk by simply passing the filename and the serialisation mode as the constructor arguments. If the serialisation mode is not specified then XML is assumed. For example:

```
nnet::NeuralNet myXMLNet( "/home/me/myNeuralNet.xml", nnet::NeuralNet::XML );
nnet::NeuralNet anotherXMLNet( "/home/me/myOtherNeuralNet.xml" );//XML is the default
nnet::NeuralNet myTextNet( "/home/me/myNeuralNet.txt", nnet::NeuralNet::PlainText );
```

Note that there is currently no error checking when loading XML nets, **if you try and load a plain text net as XML, or the file is not properly structured you will get a segmentation fault or runaway memory allocation**. This is still being looked into.

4.9 Neuron Descriptions

The output from a neuron is given by its *threshold function* which is unique to each type of neuron. This is a function of the neurons *activation value*, which is calculated the same way for each type.

The activation value a for a neuron with N inputs, i_n , each with weights w_n is given by

$$a = \sum_{n=1}^{N} i_n \times w_n + b \times w_b$$

Where b is a bias that can be assigned to a particular neuron (and w_b the bias' weight). The weights are initially random, and are then fine tuned by the training algorithms to try and give the desired output. The bias is set when the neuron is created but that process is done internally by the neuron builders. All current neuron builders set the bias to -1.

Some of the neurons have methods to change their behaviour. To get the neuron pointer to call these methods use `<tt>NeuralNet::layer(layerNumber)->neuron(neuronNumber)</tt>", where the numbers of available layers and neurons per layer can be found withNeural-Net::numberOfLayers()"` and `"NeuralNet::layer(layerNumber)->numberOfNeurons()"` respectively.

4.9.1 Linear Neuron

Linear neurons give, as the name suggests, a linear output between -1 and +1 with a gradient of $1/slopeEnd$. The value of *slopeEnd* can be set using the `LinearNeuron::setSlopeEnd(newValue)` method. If the output is greater than +slopeEnd, then the output is limited to +1; any less than -slopeEnd and the output is limited to -1. Anywhere in between gives the expected linear output of $activationvalue/slopeEnd$.

4.9.2 Sigmoid Neuron

The sigmoid neuron gives sigmoid (sort of resembles a slanted "S") output, o, of between 0 and 1 from the function

$$o = \frac{1}{1+e^{-a/r}}$$

Where r, the "response", can be set with the `SigmoidNeuron::setResponse(newValue)` method. The default is 1.

4.9.3 Tan Sigmoid Neuron

This neuron gives a similar looking output to the sigmoid neuron, but between -1 and 1. The value is given by

$$o = tanh(s \times a)$$

Where the value of s (the "scale") can be set with the `TanSigmoidNeuron::setScale(newValue)` method. The default is 1.

Author

Mark Grimes (mark.grimes@bristol.ac.uk)

5 ZVTOP

The main algorithm for ZVRES is VertexFinderClassic and for ZVKIN VertexFinderGhost

"classic" denotes implementation as in Dave Jacksons original ZVTOP paper - Nuc. Inst. Meth. A 388 (1997) 247-253

Please see the accompianing file zvtop.pdf

6 Usage tutorials

6.1 Event reconstruction required

The vertex finder and flavour tagging software expects a set of tracks - usually the tracks belonging to one jet in an event - as input. Initially, it is therefore necessary to

- obtain digitized hits
- reconstruct tracks
- run the jet finder
- reconstruct the event vertex / IP, if running the flavour tagging code
- determine the MC true jet flavour if studying flavour tag purity, efficiency
- determine the MC true quark charge to study performance of quark charge reconstruction

The performance presented at the ILC software workshop, Orsay, May 2007 was obtained with the digitization and track cheating code developed by Alexei Raspereza. For the jet finding the Durham algorithm with a y- cut of 0.04 was used, as implemented in the Satoru jet finder within MarlinReco.

An example of the steering for the first three of the above event reconstruction steps, used to obtain the shown results, is given in the steering file cheattracks+jetfind.xml. **This example assumes that the detector geometry LDC01_05Sc is used - hit collection names will need to be changed in the steering file when running with a different geometry.**

The event vertex or IP is needed for calculating the default flavour tag inputs. Since no code to do this was yet available in MarlinReco, the LCFI group implemented a procedure similar to the one used in the SGV fast MC . This should only be considered as spaceholder for a future improved procedure in which for the vertex position in the x-y-plane one would average over tracks from a number of consecutive events. The current algorithm is implemented in the PerEventIPFitterProcessor and run by calling Marlin with the steering file ipfit.xml.

The true flavour of a jet is based on the MC record in the event. It searches the event for the leading hadron, and if this is a heavy flavour particle determines which of the jets in the event is closest in angle. For heavy flavour jets, based on the leading hadron MC information also the quark charge of the heavy flavour hadron is determined. Further details can be found in the documentation of the TrueAngularJetFlavourProcessor. This part of the reconstruction is performed by running Marlin with the truejetflavour.xml steering file.

6.2 How to run the vertex finder ZVTOP

Make sure the necessary event reconstruction steps have been run in advance. The vertex finder ZVTOP has two branches: the standard branch ZVRES, based purely on topological information, and the more specialized branch ZVKIN, which uses kinematic information from heavy flavour decay chains in addition. Flavour tagging for ILC physics simulation so far has been performed using ZVRES only. The use of ZVKIN for this purpose is yet to be

explored. It should also be noted that ZVKIN parameters have not yet been tuned for ILC conditions. This vertexing algorithm is generally less tested and optimised in terms of runtime than the ZVRES branch.

For each of the two branches, a dedicated steering file is provided, named `zvres.xml` and `zvkin.xml`, respectively. The output of ZVTOP consists of one collection storing the vertices that were found, one collection holding the corresponding ReconstructedParticles decaying at these vertex positions and one collection containing one ReconstructedParticle per jet which gives access to the full decay chain. Further details on the storage of the output can be found in the LCIO Interface.

6.3 How to flavour tag jets

Before running the flavour tag, make sure the necessary event reconstruction processors and ZVTOP have been run. In the default flavour tagging algorithm information from the vertex finder is both directly used as input for the tagging neural networks and to determine, which set of neural networks is used. You can either begin by training new neural nets or use pre-trained nets. One set of nets, trained with input from the fast MC SGV, is provided with the Vertex Package. These nets are available from the new repository `tagnet` for flavour tag neural nets in the `Zeuthen CVS repository` .

Flavour tag inputs are calculated by running Marlin with the steering file `fti.xml`. As input it requires the LCIO file with information from ZVTOP and the IP fit processor (default filename zvresout.slcio). The flavour tag inputs are written out into collections of LCFloatVec's as described in more detail in the LCIO Interface.

The neural network output values are obtained from an independent Marlin processor; the corresponding example steering file is `ft.xml`. It requires the LCIO file written out in the Marlin run with `fti.xml`, which is by default called ftiout.slcio. The default algorithm is based on nine neural networks, three for each of the three classes of jets, namely those with 1, 2 and 3 or more vertices found by ZVTOP. It thus provides three output values for each jet: a b-tag value, a c-tag value for arbitrary jet sample composition and a c-tag value assuming the background is known to consist of b-jets only (yielding improved c-tag purity). These are stored as components "BTag", "CTag" and "BCTag" of an LCFloatVec collection. The collection name - FlavourTag by default - can be specified in the steering file. This collection, along with the information from the input-LCIO file, is written into an LCIO output file, named flavourtagout.slcio in the example.

6.4 How to evaluate and plot flavour tag purity vs efficiency

Flavour tag purity as function of efficiency is a measure of how well the flavour tag performs for a certain mix of jet flavours. The Vertex Package provides two processors to calculate purity and efficiency: the PlotProcessor and the LCFIAIDAPlotProcessor. An example how to call these processors is given in the steering file `ftplot.xml`.

The PlotProcessor writes out a table with the resulting efficiency and purity values as well as the cut values on the neural net outputs these correspond to, into a comma separated value file. Additionally, if the root library is linked in at compile time by defining the preprocessor flag USEROOT, the corresponding graphs are written out into a root-file. These graphs can be plotted using the root-macro MakePurityVsEfficiencyRootPlot.C provided in the macro directory. This macro also allows to plot the resulting graphs from two different runs onto the same canvas to compare performance.

The LCFIAIDAPlotProcessor provides further diagnostic tools for the flavour tag. Since different neural networks are used for the cases that 1-, 2- or at least 3 vertices are found, purity and efficiency are calculated separately for these cases. Graphs in AIDA format are created for purity vs efficiency and for the flavour leakage (i.e. the "efficiency" of tagging the wrong flavour) as function of efficiency - e.g. the leakage of usd-jets into the c-tagged sample. Also, distributions of all flavour tag input variables, the vertex charge and of the neural net output variables are created separately for the 1-, 2- and 3 or more vertex case. Optionally information can be written out into an AIDA tuple and in text format, for further details see the LCFIAIDAPlotProcessor documentation. Output from the LCFIAIDAPlotProcessor can be plotted using the python script FlavourTagInputsOverlay.py from the macro directory.

6.5 How to train new neural networks for flavour tagging

The Vertex Package is very flexible, so it is straightforward to entirely change the flavour tagging procedure. This is just an overview of changes possible, pointing out where to find further details.

In the simplest case that requires retraining, the tagging procedure itself is not changed. For example, one might want to retrain the networks after tuning ZVTOP, or changing other boundary conditions, such as track selection or composition of the input sample (as may happen when studying a specific physics channel). For this purpose, an example steering file `trainNeuralNets.xml` showing how to run the network training processor, is provided. You may choose to retrain only some of the networks - each can be enabled in the steering file independently of the others.

Changes to the tagging algorithm will require writing new processors and recompiling Marlin. A simple example would be the change of the network architecture, such as number or type of nodes and / or internal layers. Please refer to the neural network documentation for details on how networks can be defined. As long as the number and choice of input variables remains unchanged, only the training processor NeuralNetTrainerProcessor will have to be modified (or a new one added).

More complex modifications of processors are necessary when changes of the input variables are involved. This will require changes to at least three processors: the FlavourTagInputsProcessor calculating the inputs, the NeuralNetTrainerProcessor for training and the FlavourTagProcessor for obtaining the network outputs in the subsequent analysis. If further variables are to be added, this might additionally require some familiarity with the internal structure of ZVTOP. The current way of writing out the inputs into an LCFloatVec collection permits further variables to be added in a straightforward way - make sure to also update the section of the processor called at the start of the Marlin run, where the variable names are defined.

Changes to the input variables used in the training processor obviously require that the corresponding changes also be made in the processor obtaining the network outputs. In order to keep track of networks used and to allow shared use of networks within the community, a new `cvs repository tagnet` has been set up in the `Zeuthen cvs area` . **We strongly recommend submission of networks used for your studies to this repository**, along with a description of the boundary conditions for training - make sure these descriptions are as complete as possible,including details on training sample and any changes to the defaults made (track selection, ZVTOP settings, addition of input variables, if possible with a reference to the code, with which these have been obtained). Providing this information will save time when it comes to comparisons of analyses made within different frameworks / detector concepts etc.

6.6 How to determine the vertex charge

From version v00-02-02 onwards, the vertex charge is reconstructed in a dedicated processor, the VertexChargeProcessor, and stored in its own LCFloatVec collections. Two vertex charge variables are calculated, one assuming the jet is a b-jet, the other assuming it is a c-jet. Two steering files are provided: `Bvertexcharge.xml`, to be run first, producing an output LCFloatVec collection which is by default named Bcharge, and `Cvertexcharge.xml`, to be run subsequently and producing an output LCFloatVec collection with default name Ccharge.

6.7 How to apply track selection cuts

The various track selection criteria used in the code - which differ between IP determination (cf IP Fitting Cuts), ZVTOP (cf ZVTOP Cuts) and the calculation of the flavour tag inputs (cf FlavourTagInputs Cuts) - are implemented by a flexible RPCutProcessor that runs on reconstructed particles, containing the tracks in question.

7 Todo List

Class vertex_lcfi::ZVTOP::GhostFinderStage1

Upgrade to movable IP

Author

Ben Jeffery (`b.jeffery1@physics.ox.ac.uk`)

Version

0.1

Date

12/12/06

Class vertex_lcfi::ZVTOP::VertexFunctionClassic

Derivative functions not yet implemented.

Class vertex_lcfi::ZVTOP::VertexFunctionSimple

Derivative functions not yet implemented.

8 Namespace Index

8.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

9 Hierarchical Index

9.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

10 Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

11 Namespace Documentation

11.1 nnet Namespace Reference

Neural Network Namespace.

11.1.1 Detailed Description

Neural Network Namespace.

11.2 vertex_lcfi Namespace Reference

Root namespace of code used by Processors in the VertexPackage.

Namespaces

- nnet
  Neural Net namespace.
- util
  Utility classes for the vertex package.
- ZVTOP
  Namespace containing ZVTOP Implementation.

Classes

- class VertexDecaySignificance
  Calculation of Decay Significance.

- class JointProb

  Calculation of the Joint probability flavour tag inputs.
- class ParameterSignificance

  Calculation of a series of flavour tag inputs.
- class PerEventIPFitter

  Example jet variable that returns the number of tracks in the jet.
- class SecVertexProb

  Calculation of Secondary Vertex Probability.
- class TrackAttach

  Track attachment algorithm. Calculates the seed vertex and the Tracks attached to it.
- class TwoTrackPid

  Simple two track PID for gamma and Ks.
- class VertexCharge

  Calculation of the charge of the vertex.
- class VertexMass

  Calculation of the Pt corrected mass vertex flavour tag input.
- class VertexMomentum

  Calculation of Vertex Momentum.
- class VertexMultiplicity

  Calculation of the number of tracks in secondary vertices.
- class ZVKIN

  Algorithm interface for decay chain construction or vertexing.
- class ZVRES

  Algorithm interface for decay chain construction or vertexing.
- class Algo

  Algorithm interface for decay chain construction or vertexing etc.
- class DecayChain

  Decay Chain.
- class Event

  Event.
- class Jet

  Simple Jet Class.
- class ClassName

  Classname.
- class Track

  Unique Track representation.
- class TrackState

  Spatial point on a Track.
- class Vertex

  Vertex.
- class MemoryManagerType

  Base class for all MemoryManagers.
- class MetaMemoryManager

  MemoryManager Controller - see MemoryManager.
- class MemoryManager

  Memory management.

Enumerations

- enum Recalculate

  Recalculate a variable or use a chached answer if possible.

11.2.1 Detailed Description

Root namespace of code used by Processors in the VertexPackage. class TState TState.h include/TState.h author Tomas Lastovicka (LCFI) date 2007-07-26

11.3 vertex_lcfi::nnet Namespace Reference

Neural Net namespace.

11.3.1 Detailed Description

Neural Net namespace.

11.4 vertex_lcfi::util Namespace Reference

Utility classes for the vertex package.

Classes

- class HelixRep

  Multi-Perpose 3 Vector.
- class Vector3

  Multi-Perpose 3 Vector.

Enumerations

- enum Projection

  Projection Type.

Functions

- double gamSer (double a, double x)

  Incomplete gamma function P(a,x) via its series representation.
- double gamCf (double a, double x)

  Computation of the incomplete gamma function P(a,x) via its continued fraction representation.
- double gamma (double a, double x)

  Computation of the upper incomplete gamma function P(a,x)
- double lnGamma (double z)

  Computation of ln[gamma(z)] for all z>0.
- double prob (double ChiSquared, double DegreesOfFreedom)

  Calculate probability from Chi Squared and Degrees of freedom.

11.4.1 Detailed Description

Utility classes for the vertex package.

11.4.2 Function Documentation

11.4.2.1 double vertex_lcfi::util::lnGamma (double *z*)

Computation of ln[gamma(z)] for all z>0.

The algorithm is based on the article by C.Lanczos [1] as denoted in Numerical Recipes 2nd ed. on p. 207 (W.H.-Press et al.). [1] C.Lanczos, SIAM Journal of Numerical Analysis B1 (1964), 86. The accuracy of the result is better than 2e-10.

11.5 vertex_lcfi::ZVTOP Namespace Reference

Namespace containing ZVTOP Implementation.

Classes

- class CandidateVertex

  A collection of TrackState objects with a fit and vertex function maximum.
- class GaussEllipsoid

  Gaussian ellipsoid component of the vertex function.
- class GaussTube

  Gaussian tube component of the vertex function.
- class GhostFinderStage1

  First stage of ghost track alorithm - ghost finder.
- class InteractionPoint

  Interaction Point representation.
- class VertexFinderClassic

  Vertex Finding object - classic ZVTOP.
- class VertexFinderGhost

  Vertex Finding object - classic ZVTOP.
- class VertexFitter

  Vertex Fitter Interface.
- class VertexFuncMaxFinder

  Vertex Function Maximum Finder Interface.
- class VertexFuncMaxFinderClassicStepper

  Robust VertexFuncMaxFinder.
- class VertexFunction

  Vertex Function Interface.
- class VertexFunctionClassic

  VertexFunction as in ZVTOP paper.
- class VertexFunctionElement

  Vertex Fuction Element (Tubes, ellipse) Interface.
- class VertexFunctionSimple

  Simplified VertexFunction.
- class VertexResolver

  Vertex Resolver Interface.
- class VertexResolverEqualSteps

  VertexResolver as in ZVTOP paper.

11.5.1 Detailed Description

Namespace containing ZVTOP Implementation.

12 Class Documentation

12.1 vertex_lcfi::Algo< INTYPE, OUTTYPE > Class Template Reference

Algorithm interface for decay chain construction or vertexing etc.

`#include <algo.h>`

Inheritance diagram for vertex_lcfi::Algo< INTYPE, OUTTYPE >:

Public Member Functions

- virtual string name () const =0

  Name.
- virtual std::vector< string > parameterNames () const =0

  Parameter Names.
- virtual std::vector< string > parameterValues () const =0

  Parameter Values.
- virtual void setStringParameter (const string &Parameter, const string &Value)=0

  Set String Parameter.
- virtual void setDoubleParameter (const string &Parameter, const double Value)=0

  Set Double Parameter.
- virtual void setPointerParameter (const string &Parameter, void *Value)=0

  Set Pointer Parameter.
- virtual OUTTYPE calculateFor (INTYPE Input) const =0

  Run the algorithm on a jet.

12.1.1 Detailed Description

template<class INTYPE, class OUTTYPE>class vertex_lcfi::Algo< INTYPE, OUTTYPE >

Algorithm interface for decay chain construction or vertexing etc.

Description

12.1.2 Member Function Documentation

12.1.2.1 template<class INTYPE, class OUTTYPE> virtual OUTTYPE vertex_lcfi::Algo< INTYPE, OUTTYPE >::calculateFor (INTYPE *Input*) const [pure virtual]

Run the algorithm on a jet.

Calculate the Output of the Algo

Parameters

Jet	Pointer to object to be analysed

Returns

Output of the algorithm

Implemented in vertex_lcfi::ParameterSignificance, vertex_lcfi::VertexMass, vertex_lcfi::TwoTrackPid, vertex_lcfi::JointProb, vertex_lcfi::SecVertexProb, vertex_lcfi::TrackAttach, vertex_lcfi::VertexDecaySignificance, vertex_lcfi::ZVKIN, vertex_lcfi::ZVRES, vertex_lcfi::VertexCharge, vertex_lcfi::VertexMomentum, vertex_lcfi::VertexMultiplicity, and vertex_lcfi::PerEventIPFitter.

12.1.2.2 template<class INTYPE, class OUTTYPE> virtual string vertex_lcfi::Algo< INTYPE, OUTTYPE >::name ( ) const [pure virtual]

Name.

String name of the algorithm

Returns

String name

Implemented in vertex_lcfi::VertexMass, vertex_lcfi::ParameterSignificance, vertex_lcfi::TwoTrackPid, vertex_lcfi::JointProb, vertex_lcfi::TrackAttach, vertex_lcfi::SecVertexProb, vertex_lcfi::VertexDecaySignificance, vertex_lcfi::ZVKIN, vertex_lcfi::ZVRES, vertex_lcfi::VertexCharge, vertex_lcfi::VertexMomentum, vertex_lcfi::VertexMultiplicity, and vertex_lcfi::PerEventIPFitter.

12.1.2.3 template<class INTYPE, class OUTTYPE> virtual std::vector<string> vertex_lcfi::Algo< INTYPE, OUTTYPE >::parameterNames ( ) const [pure virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implemented in vertex_lcfi::VertexMass, vertex_lcfi::ParameterSignificance, vertex_lcfi::TwoTrackPid, vertex_lcfi::JointProb, vertex_lcfi::TrackAttach, vertex_lcfi::SecVertexProb, vertex_lcfi::VertexDecaySignificance, vertex_lcfi::ZVKIN, vertex_lcfi::ZVRES, vertex_lcfi::VertexCharge, vertex_lcfi::VertexMomentum, vertex_lcfi::VertexMultiplicity, and vertex_lcfi::PerEventIPFitter.

12.1.2.4 template<class INTYPE, class OUTTYPE> virtual std::vector<string> vertex_lcfi::Algo< INTYPE, OUTTYPE >::parameterValues ( ) const [pure virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implemented in vertex_lcfi::VertexMass, vertex_lcfi::ParameterSignificance, vertex_lcfi::TwoTrackPid, vertex_lcfi::JointProb, vertex_lcfi::TrackAttach, vertex_lcfi::SecVertexProb, vertex_lcfi::VertexDecaySignificance, vertex_lcfi::ZVKIN, vertex_lcfi::ZVRES, vertex_lcfi::VertexCharge, vertex_lcfi::VertexMomentum, vertex_lcfi::VertexMultiplicity, and vertex_lcfi::PerEventIPFitter.

12.1.2.5 template<class INTYPE, class OUTTYPE> virtual void vertex_lcfi::Algo< INTYPE, OUTTYPE >::setDoubleParameter (const string & *Parameter,* const double *Value*) [pure virtual]

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name

Value	double of parameter value

12.1.2.6 template<class INTYPE, class OUTTYPE> virtual void vertex_lcfi::Algo< INTYPE, OUTTYPE >::setPointerParameter (const string & *Parameter,* void * *Value*) [pure virtual]

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.1.2.7 template<class INTYPE, class OUTTYPE> virtual void vertex_lcfi::Algo< INTYPE, OUTTYPE >::setStringParameter (const string & *Parameter,* const string & *Value*) [pure virtual]

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- algo.h

12.2 vertex_lcfi::ZVTOP::CandidateVertex Class Reference

A collection of TrackState objects with a fit and vertex function maximum.

`#include <candidatevertex.h>`

Public Types

- enum eResolveType

  Type of resolution to perform - vertex position or the nearest found maximum.
- typedef VertexFitterLSM FallbackVertexFitter
- typedef VertexResolverEqualSteps FallbackVertexResolver
- typedef VertexFuncMaxFinderClassicStepper FallbackVertexFuncMaxFinder

Public Member Functions

- CandidateVertex (const std::vector< TrackState * > &Tracks, VertexFunction *VertexFunction, VertexFitter *Fitter=_getFallbackFitter(), VertexResolver *Resolver=_getFallbackResolver(), VertexFuncMaxFinder *MaxFinder=_getFallbackMaxFinder())

  Constuct with just Track list and VertexFunction.
- CandidateVertex (const std::vector< TrackState * > &Tracks, InteractionPoint *IP, VertexFunction *VertexFunction, VertexFitter *Fitter=_getFallbackFitter(), VertexResolver *Resolver=_getFallbackResolver(), VertexFuncMaxFinder *MaxFinder=_getFallbackMaxFinder())

  Constuct with Track list, InteractionPoint and VertexFunction.
- CandidateVertex (const Vector3 &Position, const Matrix3x3 &PositionError, double ChiSquaredOfFit, std::map< TrackState *, double > ChiSquaredOfTrack, double ChiSquaredOfIP)

Construct a CandidateVertex with fit information.

- CandidateVertex (const std::vector< CandidateVertex * > &Vertices, VertexFitter *Fitter=_getFallbackFitter(), VertexResolver *Resolver=_getFallbackResolver(), VertexFuncMaxFinder *MaxFinder=_getFallbackMaxFinder())

  Construct a CandidateVertex by merging other CandidateVertices.
- ~CandidateVertex ()

  Destructor.
- bool removeTrackState (TrackState *const TrackToRemove)

  Remove the first reference to a TrackState from this vertices track list.
- bool removeTrack (Track *const TrackToRemove)

  Remove the first TrackState from this vertices track list which has parent track TrackToRemove.
- void addTrackState (TrackState *TrackToAdd)

  Add a TrackState to this vertex.
- bool removeIP ()

  Remove this vertices InteractionPoint.
- void setIP (InteractionPoint *IP)

  Add an InteractionPoint.
- void mergeCandidateVertex (const CandidateVertex *SourceVertex)

  Merge another vertex into this one.
- void claimTracksFrom (const std::list< CandidateVertex * > &LosingVertices)

  Claim tracks and IP from another vertex.
- int trimByProb (const double ProbThreshold)

  Trim trackstates in order of decreasing chi squared until the vertex has a probabilty below that of the threshold.
- int trimByChi2 (const double Chi2Threshold)

  Trim trackstates with chi squared contributions bigger than the threshold.
- int trimByChi2 (const double Chi2Threshold, VertexFitter *Fitter)

  Trim trackstates with chi squared contributions bigger than the threshold.
- void invalidateFit () const

  Invalidate the fit so that the refit() is called when needed.
- void refit (bool CalculateError=0) const

  Refit the vertex.
- void refit (VertexFitter *Fitter, bool CalculateError=0) const

  Refit the vertex using the fitter specified.
- void invalidateFuncMax () const

  Invalidate the found vertex function maximum so that findVertexFuncMax() is called when needed.
- bool findVertexFuncMax () const

  Find the local vertex function maximum.
- bool isResolvedFrom (CandidateVertex *const Vertex, const double Threshold, eResolveType Type) const

  Resolve two vertices with this vertices resolver.
- bool isResolvedFrom (CandidateVertex *const Vertex, const double Threshold, eResolveType Type, VertexResolver *Resolver) const

  Resolve two vertices with a specified resolver.
- const std::vector< TrackState * > & trackStateList () const

  Return the TrackStates in this Vertex.
- bool hasTrack (Track *Track) const

  Return if this Vertex contains the passed track.
- InteractionPoint * interactionPoint () const

  Return the InteractionPoint in this Vertex.
- const Vector3 & position () const

  Return the fitted position of this Vertex.
- const Matrix3x3 & positionError () const

Return the fitted position error matrix of this Vertex.

- double distanceTo (const Vector3 &Point) const

  Return the distance from this Vertex to a point.
- double distanceTo (const CandidateVertex *const Vertex) const

  Return the distance from this Vertex to another Vertex.
- double vertexFuncMaxValue () const

  Return the value of the vertex function at this vertexes local maximum.
- const Vector3 & vertexFuncMaxPosition () const

  Return the position of this vertexes local maximum.
- double vertexFuncValue () const

  Return the value of the vertex function at the vertices position.
- double chiSquaredOfTrack (TrackState *Track) const

  Return the chi squared contribution of a trackstate in this vertex.
- double chiSquaredOfIP () const

  Return the chi squared contribution of the IP if this vertex has one.
- const std::map< TrackState *, double > & chiSquaredOfAllTracks () const

  Return the chi squared contribution of all the trackstates in this vertex.
- double chiSquaredOfFit () const

  Return the chi squared of the fit.
- double maxChiSquaredOfTrackIP () const

  Return the chi squared contribution of the Track or IP with the highest chi square contribution.

12.2.1 Detailed Description

A collection of TrackState objects with a fit and vertex function maximum.

This class is a list of TrackStates (and possibly an InteractionPoint) that are fit to a common spatial point, and which are optionally related to a maximum in the vertex function. Three external algorithm classes are used, VertexFitter,VertexResolver and VertexFuncMaxFinder. The perticular instance of these algorithms to be used by each candidate vertex is specified at construction, the constuctor defaults to fallback algorithms if none are specified.

In order to minimise the number of times a vertex is fitted, this only happens when needed. When a candidate vertex is created or its track content changed a flag is set (_FitIsValid) to 0, a function such as position() will check the status of this flag and fit the vertex if needed, so that a fit only occurs if the track content changed *and* the fit is needed. The vertex function maximum works in a similar way, except it is not currently invalidated on a change of track content as classic ZVTOP does not require this.

The per-instance algorithms can be overridden as the functions that use them are overloaded with a version that takes a pointer to the algorithm class. For example calling isResolvedFrom(Vertex, Threshold, Resolver) rather than isResolvedFrom(Vertex,Threshold). For fitting and max finding this is slightly complicated by the caching system mentioned above. For example to use a different fitter than that specified at constuction to get the position of the vertex, one must call refit(Fitter) and then position().

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

12.2.2 Member Typedef Documentation

12.2.2.1 typedef VertexFitterLSM vertex_lcfi::ZVTOP::CandidateVertex::FallbackVertexFitter

This typedef determines the VertexFitter used if none is specified in the CandidateVertex constuctor. If changed you may need to change the includes in the cpp file

12.2.2.2 typedef VertexFuncMaxFinderClassicStepper vertex_lcfi::ZVTOP::CandidateVertex::FallbackVertexFuncMaxFinder

This typedef determines the VertexFuncMaxFinder used if none is specified in the CandidateVertex constuctor. If changed you may need to change the includes in the cpp file

12.2.2.3 typedef VertexResolverEqualSteps vertex_lcfi::ZVTOP::CandidateVertex::FallbackVertexResolver

This typedef determines the VertexResolver used if none is specified in the CandidateVertex constuctor. If changed you may need to change the includes in the cpp file

12.2.3 Constructor & Destructor Documentation

12.2.3.1 vertex_lcfi::ZVTOP::CandidateVertex::CandidateVertex (const std::vector< TrackState * > & *Tracks*, VertexFunction * *VertexFunction*, VertexFitter * *Fitter* = `_getFallbackFitter()`, VertexResolver * *Resolver* = `_getFallbackResolver()`, VertexFuncMaxFinder * *MaxFinder* = `_getFallbackMaxFinder()`)

Constuct with just Track list and VertexFunction.

Creates a CandidateVertex from a set of TrackState objects, a VertexFunction but no InteractionPoint. As no fit or VertexFunction maximum is specifed at constuction the flags for these are set to invalid so that they are worked out when needed. Note defaults for fitter, resolver and max finder.

Parameters

Tracks	A vector of pointers to the TrackState objects that form this CandidateVertex. The same TrackState can be given to multiple CandidateVertex objects, but this is may not desierable as the TrackState would then be swum back and forth between the vertices.
VertexFunction	Pointer to the function which is explored for the nearest maxima to the fit vertex.
Fitter	Pointer to the VertexFitter that the vertex uses to fit itself. Defaults to the typedef FallbackVertexFitter.
Resolver	Pointer to the VertexResolver that the vertex uses to resolve itself from others. Defaults to the typedef FallbackVertexResolver.
MaxFinder	Pointer to the VertexFuncMaxFinder that the vertex uses to find the nearest VertexFunction maximum. Defaults to the typedef FallbackVertexFuncMaxFinder.

12.2.3.2 vertex_lcfi::ZVTOP::CandidateVertex::CandidateVertex (const std::vector< TrackState * > & *Tracks*, InteractionPoint * *IP*, VertexFunction * *VertexFunction*, VertexFitter * *Fitter* = `_getFallbackFitter()`, VertexResolver * *Resolver* = `_getFallbackResolver()`, VertexFuncMaxFinder * *MaxFinder* = `_getFallbackMaxFinder()`)

Constuct with Track list, InteractionPoint and VertexFunction.

Creates a CandidateVertex from a set of TrackState objects, a VertexFunction and an InteractionPoint. As no fit or VertexFunction maximum is specifed at constuction the flags for these are set to invalid so that they are worked out when needed. Note defaults for fitter, resolver and max finder.

Parameters

Tracks	A vector of pointers to the TrackState objects that form this CandidateVertex. The same TrackState can be given to multiple CandidateVertex objects, but this is may not desierable as the TrackState would then be swum back and forth between the vertices.
IP	A pointer to the InteractionPoint associated with the vertex
VertexFunction	Pointer to the function which is explored for the nearest maxima to the fit vertex.
Fitter	Pointer to the VertexFitter that the vertex uses to fit itself. Defaults to the typedef FallbackVertexFitter.
Resolver	Pointer to the VertexResolver that the vertex uses to resolve itself from others. Defaults to the typedef FallbackVertexResolver.
MaxFinder	Pointer to the VertexFuncMaxFinder that the vertex uses to find the nearest VertexFunction maximum. Defaults to the typedef FallbackVertexFuncMaxFinder.

12.2.3.3 vertex_lcfi::ZVTOP::CandidateVertex::CandidateVertex (const Vector3 & *Position*, const Matrix3x3 & *PositionError*, double *ChiSquaredOfFit*, std::map< TrackState *, double > *ChiSquaredOfTrack*, double *ChiSquaredOfIP*)

Construct a CandidateVertex with fit information.

Creates a CandidateVertex that contains fit information and in which the fit flag is set valid.

Parameters

Position	Position of the fitted vertex
PositionError	ErrorMatrix of the fitted vertex
ChiSquaredOfFit	Chi Squared of total fit
ChiSquaredOf-Track	A std::map of track chi squareds indexed by trackstate pointer
ChiSquaredOfIP	Chi squared contribution of IP object (if any)

12.2.3.4 vertex_lcfi::ZVTOP::CandidateVertex::CandidateVertex (const std::vector< CandidateVertex * > & *Vertices*, VertexFitter * *Fitter* = `_getFallbackFitter()`, VertexResolver * *Resolver* = `_getFallbackResolver()`, VertexFuncMaxFinder * *MaxFinder* = `_getFallbackMaxFinder()`)

Construct a CandidateVertex by merging other CandidateVertices.

Creates a CandidateVertex by merging the tracks and IP from a collection of CV's track duplicates are removed by checking if TrackStates have the same parent, if there is more than one IP object the IP from the last vertex in the list is used. If the list is empty you get an empty vertex! Ignores vertex functions.

Parameters

Vertices	Position of the fitted vertex
Fitter	Pointer to the VertexFitter that the vertex uses to fit itself. Defaults to the typedef FallbackVertexFitter.
Resolver	Pointer to the VertexResolver that the vertex uses to resolve itself from others. Defaults to the typedef FallbackVertexResolver.
MaxFinder	Pointer to the VertexFuncMaxFinder that the vertex uses to find the nearest VertexFunction maximum. Defaults to the typedef FallbackVertexFuncMaxFinder.

12.2.3.5 vertex_lcfi::ZVTOP::CandidateVertex::~CandidateVertex ( ) [inline]

Destructor.

Does not delete any objects passed at construction.

12.2.4 Member Function Documentation

12.2.4.1 void vertex_lcfi::ZVTOP::CandidateVertex::addTrackState (TrackState * *TrackToAdd*)

Add a TrackState to this vertex.

Adds the TrackState to this Vertices track list, and invalidates the fit via invalidateFit().

Parameters

TrackToAdd	Pointer to the TrackState to add.

12.2.4.2 const std::map<TrackState*, double>& vertex_lcfi::ZVTOP::CandidateVertex::chiSquaredOfAllTracks ( ) const

Return the chi squared contribution of all the trackstates in this vertex.

Note this may cause the vertex to be fit if needed.

Returns

A std::map of chi squared vlaues with TrackState pointers as the key

12.2.4.3 double vertex_lcfi::ZVTOP::CandidateVertex::chiSquaredOfFit ( ) const

Return the chi squared of the fit.

Note this may cause the vertex to be fit if needed.

Returns

The chi squared value.

12.2.4.4 double vertex_lcfi::ZVTOP::CandidateVertex::chiSquaredOfIP ( ) const

Return the chi squared contribution of the IP if this vertex has one.

Note this may cause the vertex to be fit if needed.

Returns

Chi squared of the ip if this vertex has one zero otherwise

12.2.4.5 double vertex_lcfi::ZVTOP::CandidateVertex::chiSquaredOfTrack (TrackState * *Track*) const

Return the chi squared contribution of a trackstate in this vertex.

If the trackstate is found its chi squared is returned. Note this may cause the vertex to be fit if needed.

Returns

The chi squared of the track if found, -1 otherwise.

12.2.4.6 void vertex_lcfi::ZVTOP::CandidateVertex::claimTracksFrom (const std::list< CandidateVertex * > & *LosingVertices*)

Claim tracks and IP from another vertex.

Message all the vertices in the list LosingVertices to remove trackstates that have the same parent as the trackstates in this vertex. The fits of the losing vertices are automatically invalidated as we call removeTrack(). Also removes the IP from losing vertices if we have it

Parameters

LosingVertices	A vector of pointers to the vertices that lose the tracks we have.

12.2.4.7 double vertex_lcfi::ZVTOP::CandidateVertex::distanceTo (const Vector3 & *Point*) const

Return the distance from this Vertex to a point.

Note this may lead to this vertex being refit as position() is called.

Parameters

Point	Vector3 of point.

Returns

Distance to point.

12.2.4.8 double vertex_lcfi::ZVTOP::CandidateVertex::distanceTo (const CandidateVertex *const *Vertex*) const

Return the distance from this Vertex to another Vertex.

Note this may lead to either vertex being refit as position() is called.

Parameters

Vertex	to measure distance to.

Returns

Distance to Vertex.

12.2.4.9 bool vertex_lcfi::ZVTOP::CandidateVertex::findVertexFuncMax ( ) const

Find the local vertex function maximum.

Find the nearest vertex function maximum using the vertexes _MaxFinder, using the fitted position of this vertex as the inital guess. Note this may lead to this vertex being refit as position() is called.

12.2.4.10 bool vertex_lcfi::ZVTOP::CandidateVertex::hasTrack (Track * *Track*) const

Return if this Vertex contains the passed track.

Parameters

Track	Pointer to the query track

Returns

true if this vertex has a trackstate with this Track as parent

12.2.4.11 void vertex_lcfi::ZVTOP::CandidateVertex::invalidateFit ( ) const

Invalidate the fit so that the refit() is called when needed.

Called by any function that modifies the vertex so that it is refit before being used again.

Note: Currently does not invalidate the Vertex Function Maximum as this stays constant through a Vertices life for classic ZVTOP Invalidation of Vertex Func Max should be a parameter

12.2.4.12 bool vertex_lcfi::ZVTOP::CandidateVertex::isResolvedFrom (CandidateVertex *const *Vertex*, const double *Threshold*, eResolveType *Type*) const

Resolve two vertices with this vertices resolver.

Uses the VertexResolver stored in _Resolver to resolve this vertex and the one specified.

Parameters

Vertex	Vertex to resolve this one with.
Threshold	Threshold for resolution, see VertexResolverEqualSteps for default resolver.
Type	Point to use for resolution, either FittedPosition or NearestMaximum

Returns

1 if the vertices are resolved, 0 otherwise.

12.2.4.13 bool vertex_lcfi::ZVTOP::CandidateVertex::isResolvedFrom (CandidateVertex *const *Vertex, const double *Threshold, eResolveType *Type, VertexResolver * *Resolver*) const

Resolve two vertices with a specified resolver.

Uses the VertexResolver specified to resolve this vertex and the one specified.

Parameters

Vertex	Vertex to resolve this one with.
Threshold	Threshold for resolution, implementation depends on resolver.
Type	Point to use for resolution, either FittedPosition or NearestMaximum
Resolver	Pointer to VertexResolver to use.

Returns

1 if the vertices are resolved, 0 otherwise.

12.2.4.14 double vertex_lcfi::ZVTOP::CandidateVertex::maxChiSquaredOfTrackIP ( ) const

Return the chi squared contribution of the Track or IP with the highest chi square contribution.

Note this may cause the vertex to be fit if needed.

Returns

The chi squared contibution.

12.2.4.15 void vertex_lcfi::ZVTOP::CandidateVertex::mergeCandidateVertex (const CandidateVertex * *SourceVertex*)

Merge another vertex into this one.

Adds all the trackstates and the IP if held from SourceVertex to this vertices trackstate list, after removing any trackstates which have the same parent to avoid duplicates. The fit is automatically invalidated as this function calls addTrackState() If the vertex function maximum of the source vertex is bigger than this vertices, then it is also copied to this vertex.

SourceVertex remains unchanged.

Parameters

SourceVertex	The vertex to be merged into this one.

12.2.4.16 const Vector3& vertex_lcfi::ZVTOP::CandidateVertex::position ( ) const

Return the fitted position of this Vertex.

If the fit of this vertex is invalid it is refit and then the position returned.

Returns

A Vector3 of the fitted position.

12.2.4.17 const Matrix3x3& vertex_lcfi::ZVTOP::CandidateVertex::positionError ( ) const

Return the fitted position error matrix of this Vertex.

If the fit of this vertex is invalid it is refit using refit() and then the position error returned.

Returns

A Matrix3x3 of the fitted position error.

12.2.4.18 void vertex_lcfi::ZVTOP::CandidateVertex::refit (bool *CalculateError* = 0) const

Refit the vertex.

Used by methods that need fit information such as position() if the fit is flagged as invalid.

Parameters

CalculateError	If true the vertex error is calculated - a time saving device

12.2.4.19 void vertex_lcfi::ZVTOP::CandidateVertex::refit (VertexFitter * *Fitter,* bool *CalculateError* = 0) const

Refit the vertex using the fitter specified.

Fits the vertex with the one specified and flags the fit as valid.

Parameters

Fitter	Fitter to use.
CalculateError	If true the vertex error is calculated - a time saving device

12.2.4.20 bool vertex_lcfi::ZVTOP::CandidateVertex::removeIP ( )

Remove this vertices InteractionPoint.

If this vertex had a pointer to an IP it is removed, and the fit invalidated via invalidateFit().

Returns

1 if this vertex had an IP; 0 otherwise.

12.2.4.21 bool vertex_lcfi::ZVTOP::CandidateVertex::removeTrack (Track *const *TrackToRemove*)

Remove the first TrackState from this vertices track list which has parent track TrackToRemove.

This vertices TrackState list is searched for the first TrackState that has TrackToRemove as its ParentTrack. If found it is removed and the fit invalidated via invalidateFit().

Parameters

TrackToRemove	Pointer to the TrackState to remove.

Returns

1 if a track was removed; 0 otherwise.

12.2.4.22 bool vertex_lcfi::ZVTOP::CandidateVertex::removeTrackState (TrackState *const *TrackToRemove*)

Remove the first reference to a TrackState from this vertices track list.

This vertices TrackState list is searched for the first reference to TrackToRemove. If found it is removed and the fit invalidated via invalidateFit().

Parameters

TrackToRemove	Pointer to the TrackState to remove.

Returns

1 if a TrackState was removed; 0 otherwise.

12.2.4.23 void vertex_lcfi::ZVTOP::CandidateVertex::setIP (InteractionPoint * *IP*)

Add an InteractionPoint.

Sets the IP of this vertex, replacing the current one, if any. The fit is then invalidated via invalidateFit().

Parameters

IP	Pointer to the InteractionPoint to set.

12.2.4.24 const std::vector<TrackState*>& vertex_lcfi::ZVTOP::CandidateVertex::trackStateList ( ) const [inline]

Return the TrackStates in this Vertex.

Returns

A Vector of pointers to the TrackStates in the vertex

12.2.4.25 int vertex_lcfi::ZVTOP::CandidateVertex::trimByChi2 (const double *Chi2Threshold*)

Trim trackstates with chi squared contributions bigger than the threshold.

Using the fitter of this vertex (specified at constuction or default) the trackstate with the highest chi squared is removed if it is above threshold. If one was removed then refit and check again, repeating until threshold is reached or we only have one track left.. Does not effect the IP held by this vertex if any.

Parameters

Chi2Threshold	Threshold for removal.

Returns

Number of trackstates removed.

12.2.4.26 int vertex_lcfi::ZVTOP::CandidateVertex::trimByChi2 (const double *Chi2Threshold,* VertexFitter * *Fitter*)

Trim trackstates with chi squared contributions bigger than the threshold.

Using the fitter specified, the trackstate with the highest chi squared is removed if it is above threshold. If one was removed then refit and check again, repeating until one is not removed. Does not effect the IP held by this vertex if any.

Parameters

Chi2Threshold	Threshold for removal.
Fitter	VertexFitter to use

Returns

Number of trackstates removed.

12.2.4.27 int vertex_lcfi::ZVTOP::CandidateVertex::trimByProb (const double *ProbThreshold*)

Trim trackstates in order of decreasing chi squared until the vertex has a probabilty below that of the threshold.

Using the fitter of this vertex (specified at constuction or default) the trackstate with the highest chi squared is removed if it is below threshold. If one was removed then refit and check again, repeating until we have reached thresold.

Parameters

ProbThreshold	Threshold for removal.

Returns

Number of trackstates removed.

12.2.4.28 const Vector3& vertex_lcfi::ZVTOP::CandidateVertex::vertexFuncMaxPosition ( ) const

Return the position of this vertexes local maximum.

If the position of this vertex is invalid it is refound using findVertexFuncMax() and then returned.

Returns

The maximum's position.

12.2.4.29 double vertex_lcfi::ZVTOP::CandidateVertex::vertexFuncMaxValue ( ) const

Return the value of the vertex function at this vertexes local maximum.

If the maximum of this vertex is invalid it is refound using findVertexFuncMax() and then the value returned.

Returns

The maximum's value.

12.2.4.30 double vertex_lcfi::ZVTOP::CandidateVertex::vertexFuncValue ( ) const

Return the value of the vertex function at the vertices position.

Note that this return the value at the fitted position, not the nearest maximum

Returns

The vertex function valuer

The documentation for this class was generated from the following file:

- candidatevertex.h

12.3 vertex_lcfi::ClassName Class Reference

Classname.

`#include <template.h>`

Public Member Functions

- function (params)
  Short description.

12.3.1 Detailed Description

Classname.

Description

12.3.2 Member Function Documentation

12.3.2.1 vertex_lcfi::ClassName::function (params)

Short description.

Description

Parameters

Parameter	

Returns

Return

The documentation for this class was generated from the following file:

- template.h

12.4 vertex_lcfi::DecayChain Class Reference

Decay Chain.

`#include <decaychain.h>`

Public Member Functions

- DecayChain ()
  Default Constructor.
- DecayChain (const DecayChain &OldDecayChain)
  Copy Constructor.
- DecayChain (Jet *MyJet, const std::vector< Track * > &Tracks, const std::vector< Vertex * > &Vertices)
  Full Constructor.
- Jet * jet () const
  Owner Jet.
- const std::vector< Track * > & allTracks () const
  All tracks contained in DecayChain.
- const std::vector< Track * > & attachedTracks () const
  Attached tracks contained in DecayChain.
- const std::vector< Vertex * > & vertices () const
  Vertices contained in DecayChain.
- double charge () const
  Charge sum of all tracks in the DecayChain.
- const util::Vector3 & momentum () const
  Perigee momentum sum of all tracks in the DecayChain.
- void addTrack (Track *Track)
  Add Track.
- bool removeTrack (Track *Track)
  Remove Track.
- bool hasTrack (Track *Track) const
  Does the DecayChain have this Track?
- void addVertex (Vertex *Vertex)
  Add Vertex.

- bool removeVertex (Vertex *Vertex)

  Remove Vertex.
- bool hasVertex (Vertex *Vertex) const

  Does the DecayChain have this Vertex?

12.4.1 Detailed Description

Decay Chain.

Description

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

12.4.2 Constructor & Destructor Documentation

12.4.2.1 vertex_lcfi::DecayChain::DecayChain (Jet * *MyJet,* const std::vector< Track * > & *Tracks,* const std::vector< Vertex * > & *Vertices*)

Full Constructor.

Parameters

Tracks	A vector of pointers to tracks that are attahced to the decay chain
Vertices	A vector of pointers to vertices that are contained in the decay chain

12.4.3 Member Function Documentation

12.4.3.1 void vertex_lcfi::DecayChain::addTrack (Track * *Track*)

Add Track.

Add a track to the DecayChain as an attached track

Parameters

Track	Pointer to the track to be added

12.4.3.2 void vertex_lcfi::DecayChain::addVertex (Vertex * *Vertex*)

Add Vertex.

Add a Vertex to the DecayChain

Parameters

Vertex	Pointer to the Vertex to be added

12.4.3.3 const std::vector<Track*>& vertex_lcfi::DecayChain::allTracks ( ) const

All tracks contained in DecayChain.

Returns all tracks contained in both attached tracks and in vertices, in order of increasing d0

Returns

A vector of pointers to all tracks in the decay chain

12.4.3.4 const std::vector<Track*>& vertex_lcfi::DecayChain::attachedTracks ( ) const

Attached tracks contained in DecayChain.

Returns attached tracks, in order of increasing d0

Returns

A vector of pointers to attached tracks in the decay chain

12.4.3.5 double vertex_lcfi::DecayChain::charge ( ) const

Charge sum of all tracks in the DecayChain.

Sums charge of all tracks in vertices and attached

Returns

double of sum charge

12.4.3.6 bool vertex_lcfi::DecayChain::hasTrack (Track * *Track*) const

Does the DecayChain have this Track?

Does this DecayChain have a perticular track?

Parameters

Track	Pointer to track to be checked

Returns

1 if track was found, 0 if not found

12.4.3.7 bool vertex_lcfi::DecayChain::hasVertex (Vertex * *Vertex*) const

Does the DecayChain have this Vertex?

Does this DecayChain have a perticular Vertex?

Parameters

Vertex	Pointer to Vertex to be checked

Returns

1 if Vertex was found, 0 if not found

12.4.3.8 Jet* vertex_lcfi::DecayChain::jet ( ) const [inline]

Owner Jet.

Returns a pointer to the Jet that contains this decay chain

Returns

A vector of pointers to all tracks in the decay chain

12.4.3.9 const util::Vector3& vertex_lcfi::DecayChain::momentum ( ) const

Perigee momentum sum of all tracks in the DecayChain.

Sums momentum of all tracks in vertices and attached tracks

Returns

Vector3 of momentum

12.4.3.10 bool vertex_lcfi::DecayChain::removeTrack (Track * *Track*)

Remove Track.

Remove a track from the DecayChain, track will be removed from both attached tracks and vertices

Parameters

Track	Pointer to track to be removed

Returns

1 if track was found and removed, 0 if not found

12.4.3.11 bool vertex_lcfi::DecayChain::removeVertex (Vertex * *Vertex*)

Remove Vertex.

Remove a Vertex from the DecayChain

Parameters

Vertex	Pointer to Vertex to be removed

Returns

1 if Vertex was found and removed, 0 if not found

12.4.3.12 const std::vector<Vertex*>& vertex_lcfi::DecayChain::vertices ( ) const

Vertices contained in DecayChain.

Returns vertices, in order of increasing rphi radius from the origin

Returns

A vector of pointers to vertices in the decay chain

The documentation for this class was generated from the following file:

- decaychain.h

12.5 DSTPlotProcessor Class Reference

Creates some sample plots from the data calculated by the LCFI vertex package.

`#include <DSTPlotProcessor.h>`

Inherits Processor.

12.5.1 Detailed Description

Creates some sample plots from the data calculated by the LCFI vertex package.

An example of getting the flavour tag results from the LCIO file and plotting an efficiency purity graph with them. Also plots a graph of jet energies for good measure.

The processor checks the specifed LCFloatVec collections for the flavour tag values "BTag", "CTag" and "BCTag" which are the names that FlavourTagProcessor stores its b tag, c tag and c tag (only b background) values in respectively.

These values are checked against the true jet flavour (from the TrueJetFlavour LCIntVec) and efficiency-purity values calculated for a range of cuts.

The jet energy is taken from the energy of the reconstructed particle used to represent the jet.

Getting Root output

To output to a Root file instead of CSV files the processor has to be compiled with the USEROOT preprocessor flag defined. You could add "#define USEROOT" to the code, or more easily add the line

USERINCLUDES += -D USEROOT

to the userlib.gmk file that is in the Marlin directory. If Marlin is not already set up to use Root then you will also need to add the following lines (this assumes a fully working root installation):

USERINCLUDES += `root-config --cflags`

USERLIBS += `root-config --libs`

Input

From the LCIO file, flavour tag variable values of:

"BTag" "CTag" "BCTag"

And

"JetType"

Output

If the USEROOT preprocessor flag was defined when this processor was compiled, then the output will be a root file with the filename specified in the steering file. Otherwise, the efficiency-purity values will be output as comma separated values to the file <filename>+".csv", and the jet energies to <filename>+"-JetEnergies.csv".

Parameters

JetCollection-Name	Name of the ReconstructedParticle collection that represents jets.
FlavourTag-Collections	Names of the LCFloatVec collections holding the Flavour tags, all tags in this list will be produced in one file for comparison
TrueJetFlavour-Collection	LCIntVec that contains the MC Jet flavour (from TrueJetFlavourProcessor)
OutputFilename	The name of the file that will hold the output.

The documentation for this class was generated from the following file:

- DSTPlotProcessor.h

12.6 vertex_lcfi::Event Class Reference

Event.

`#include <event.h>`

Public Member Functions

- Event ()

  Default Constructor.
- Event (const Vector3 &Position, const SymMatrix3x3 &Error)

Construct with interaction point.

- Event (Vertex *ipVertex)

  Construct with interaction point as vertex.

- void addJet (Jet *Jet)

  Add Jet.

- const std::vector< Jet * > & jets () const

  Get Jets.

- void addTrack (Track *Track)

  Add Track.

- const std::vector< Track * > & tracks () const

  Get Tracks.

- void replacePrimaryVertex (Vertex *NewPrimary)

  Replace Primary Vertex.

- const Vector3 & interactionPoint () const

  Interaction Point position.

- const SymMatrix3x3 & interactionPointError () const

  Interaction Point position error.

- Vertex * ipVertex () const

  Event Vertex.

12.6.1 Detailed Description

Event.

Description

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

12.6.2 Constructor & Destructor Documentation

12.6.2.1 vertex_lcfi::Event::Event ( )

Default Constructor.

Sets ip to origin and ip error to 10micron spherical

12.6.2.2 vertex_lcfi::Event::Event (const Vector3 & *Position*, const SymMatrix3x3 & *Error*)

Construct with interaction point.

Parameters

Position	ip position
Error	ip error

12.6.2.3 vertex_lcfi::Event::Event (Vertex * *ipVertex*)

Construct with interaction point as vertex.

Parameters

Pointer	to IP Vertex

12.6.3 Member Function Documentation

12.6.3.1 void vertex_lcfi::Event::addJet (Jet * *Jet*)

Add Jet.

Parameters

Jet	Pointer to jet

12.6.3.2 void vertex_lcfi::Event::addTrack (Track * *Track*)

Add Track.

Parameters

Track	Pointer to track

12.6.3.3 const Vector3& vertex_lcfi::Event::interactionPoint ( ) const

Interaction Point position.

Returns

Vector3 of interaction point position

12.6.3.4 const SymMatrix3x3& vertex_lcfi::Event::interactionPointError ( ) const

Interaction Point position error.

Returns

SymMatrix3x3 of interaction point position error

12.6.3.5 Vertex* vertex_lcfi::Event::ipVertex ( ) const

Event Vertex.

Returns

Pointer to this events primary vertex (IP)

12.6.3.6 const std::vector<Jet*>& vertex_lcfi::Event::jets ( ) const

Get Jets.

Returns

Vector of pointers to jets in the event

12.6.3.7 void vertex_lcfi::Event::replacePrimaryVertex (Vertex * *NewPrimary*) [inline]

Replace Primary Vertex.

Replace the current Primary Vertex

12.6.3.8 const std::vector<Track*>& vertex_lcfi::Event::tracks ( ) const

Get Tracks.

Returns

Vector of pointers to tracks in the event

The documentation for this class was generated from the following file:

- event.h

12.7 FlavourTagInputsProcessor Class Reference

Calculates the Flavour tag input variables for flavour tagging.

`#include <FlavourTagInputsProcessor.h>`

Inherits Processor.

Collaboration diagram for FlavourTagInputsProcessor:

12.7.1 Detailed Description

Calculates the Flavour tag input variables for flavour tagging.

The aim of the processor is to calculate a series of highly discriminating tagging variables. At present the default variables are the one defined in the R. Hawking LC note LC-PHSM-2000-021. All the variables are calculated inside independent classes that inherit from the vertex_lcfi::Algo template and not in the main processor file. This makes the processor file extremely flexible and new variables easy to add. Similary it is also very simple to remove undesired variables. The following variables are presently calculated (variables depending on the vertex_lcfi::TrackAttach procedure are marked by $*$, variables depending on vertex_lcfi::TwoTrackPid are marked by $\hat{}$)

D0Significance1 - calculated in vertex_lcfi::ParameterSignificance

D0Significance2 - calculated in vertex_lcfi::ParameterSignificance

Z0Significance1 - calculated in vertex_lcfi::ParameterSignificance

Z0Significance2 - calculated in vertex_lcfi::ParameterSignificance

Momentum1 - calculated in vertex_lcfi::ParameterSignificance

Momentum2 - calculated in vertex_lcfi::ParameterSignificance

JointProbRPhi - $\hat{}$ calculated in vertex_lcfi::JointProb

JointProbZ - $\hat{}$ calculated in vertex_lcfi::JointProb

DecayLengthSignificance - calculated in vertex_lcfi::VertexDecaySignificance

DecayLength - calculated in vertex_lcfi::VertexDecaySignificance

PTCorrectedMass - $*$ calculated in vertex_lcfi::VertexMass

RawMomentum - $*$ calculated in vertex_lcfi::VertexMomentum

NumTracksInVertices - calculated in vertex_lcfi::VertexMultiplicity

SecondaryVertexProbability - $*$ calculated in vertex_lcfi::SecVertexProb

For more information about the algorithms themselves please consult the specific algorithm documentation pages. The processor also uses the following algorithms:

vertex_lcfi::TwoTrackPid - algorithm that calulates the id of two charged tracks by using mass considerations. This algorithm removes tracks consistent with the hypothesis that they have been generated from Ks and gamma. This algorithm is not used in ZVTOPZVRESProcessor or ZVTOPZVKINProcessor

vertex_lcfi::TrackAttach - algorithm that adds tracks close to the seed vertex.

Input

- A collection of ReconstructedParticles that represents the jets in the event (obtained from a jet finder, say SatoruJetFinderProcessor).
- A collection of vertices that contains the per event primary vertices; one for each event. (optional) This collection is filled in the vertex_lcfi::PerEventIPFitter processor.
- A collection of decay chains as filled by the the ZVTOPZVRESProcessor or ZVTOPZVKINProcessor.

Output

The processor writes into the selected lcio output file. All the values calculated by the processor are saved in the same LCFloatVec collection. The default name of the output collection is FlavourTagInputs.For more details see the interface documentation.

Parameters

JetRPCollection	Name of the ReconstructedParticle collection that represents jets.
IPVertex-Collection	Name of the Vertex collection that contains the primary vertices (optional)
FlavourTag-InputsCollection	Name of the LCFloatVec Collection that will be created to contain the flavour tag inputs

The following parameters are parameters for the algorithms used by the processor.These parameters are all optional.

Parameters

LayersHit	Momentum cuts will be applied on number of LayersHit and LayersHit minus one, used by vertex_lcfi::ParameterSignificance
AllLayers-MomentumCut	Cut on the minimum momentum if track hits LayersHit, used by vertex_lcfi::Parameter-Significance
AllbutOne-Layers-MomentumCut	Cut on the minimum momentum if track hits LayersHit minus one, used by vertex_lcfi::-ParameterSignificance
JProbMaxD0-Significance	Maximum d0 significance of tracks used to calculate the joint probability, used in vertex_lcfi-::JointProb
JProbMax-D0andZ0	Maximum d0 and z0 of tracks used to calculate the joint probability, used in vertex_lcfi::Joint-Prob
PIDChi2Cut	Cut on the Chi squared of two tracks being in the same vertex, used by vertex_lcfi::TwoTrack-Pid
PIDMaxGamma-Mass	Cut on the upper limit of the photon candidate mass, used by vertex_lcfi::TwoTrackPid
PIDMaxKsMass	Cut on the upper limit of the Ks candidate mass, used by vertex_lcfi::TwoTrackPid
PIDMinKsMass	Cut on the lower limit of the Ks candidate mass, used by vertex_lcfi::TwoTrackPid
PIDRPhiCut	Cut on the maximum RPhi of the Ks/gamma decay vertex candidate, used by vertex_lcfi::-TwoTrackPid
PIDSignificance-Cut	Cut on the minimum RPhi significance of the tracks, used by vertex_lcfi::TwoTrackPid
SecondVertex-Ntrackscut	Cut on the minimum number of tracks in the seed vertex, used by vertex_lcfi::SecVertexProb
SecondVertex-Prob-Chisquarecut	Cut on the Chi Squared of the seed vertex, used by vertex_lcfi::SecVertexProb
TrackAttachAll-Secondary-Tracks	include or exclude tracks in the inner vertices for the track attachment.
TrackAttach-Closeapproach-Cut	upper cut on track distance of closest approach to the seed axis used by vertex_lcfi::Track-Attach (when used for $*$ flagged variables)
TrackAttachLoD-Cutmax	Cut determining the maximum L/D for the track attachment, used by vertex_lcfi::TrackAttach (when used for $*$ flagged variables)

TrackAttachLoD-Cutmin	Cut determining the minimum L/D for the track attachment, used by vertex_lcfi::TrackAttach (when used for $*$ flagged variables)
VertexMassMax-Kinematic-CorrectionSigma	Maximum Sigma (based on error matrix) by which the vertex axis can move when kinematic correction is applied, used by vertex_lcfi::VertexMass
VertexMassMax-Momentum-AngleCut	Upper cut on angle between momentum of vertex and the vertex axis, used by vertex_lcfi::-VertexMass
VertexMassMax-Momentum-Correction	Maximum factor, by which vertex mass can be corrected, used by vertex_lcfi::VertexMass

As a final remark one should notice that two additional values are stored in the Output LC Collection. These are:

NumVertices - number of vertices in the jet; used to determine what variables to use in the following flavour tag processor. Calculated in the processor.

DecayLength(SeedToIP)- distance from the vertex seed in the trackattach processor and IP. This veriable can be used for further analysis, but it is not used in flavour tagging. Calculated in the processor.

Author

Erik Devetak(erik.devetak1@physics.ox.ac.uk),
interface by Ben Jeffery (ben.jeffery1@physics.ox.ac.uk)

The documentation for this class was generated from the following file:

- FlavourTagInputsProcessor.h

12.8 FlavourTagProcessor Class Reference

Performs a neural net based flavour tag using data calculated by the LCFI vertex package.

`#include <FlavourTag.h>`

Inherits Processor.

12.8.1 Detailed Description

Performs a neural net based flavour tag using data calculated by the LCFI vertex package.

Loads in previously trained neural networks from the filenames provided in the steering file, and performs a flavour tag with them using the data previously stored in the file by the FlavourTagInputsProcessor. The networks can be trained using the NeuralNetTrainerProcessor.

This processor requires 9 neural networks, which are 3 for each of the 1 vertex, 2 vertices and 3 or more vertices cases. These 3 are a b jet tagging network, a c jet tagging network and a c jet with only b background tagging network. If any of these saved neural network files are not present the processor will throw a lcio::Exception. The networks can be in either text or XML format; the processor checks to see if the file starts with "<?xml" and decides whether to load as text or XML.

N.B. The code that loads the XML networks is currently a little shaky. **If the XML is not properly formed then you may get a segmentation fault or runaway memory allocation leading to Marlin crashing.** This is still being looked into.

For more information on the tagging variables used as input, have a look at the documentation for FlavourTagInputs-Processor. The flavour tag result will be in the range 0 to 1; so to select tagged jets apply a cut on this value (e.g. the b-tag value to tag b-jets). If anything goes wrong (that doesn't produce an exception) then a -1 will be stored instead.

Input

- 9 previously trained neural networks (trained by NeuralNetTrainerProcessor).

- A collection of LCFloatVec that hold the flavour tag variables (put in the lcio file by FlavourTagInputs-Processor).
- A collection of ReconstructedParticles that represents the jets in the event (put in by your jet finder, say SatoruJetFinderProcessor).

Output

- A collection of LCFloatVec that contains the 3 tag results for each jet (b tag, c tag and c only b background tag). The collection will have a float vector for each jet in the same order as the jets; so for example, the tags for "pJetCollection->getElementAt(3)" will be in "pFlavourTagCollection->getElementAt(3)". For more details see the interface documentation

Parameters

JetCollection-Name	The name of the collection of ReconstructedParticles representing the jets.
FlavourTag-InputsCollection	The name of the collection of LCFloatVec that is the flavour tag inputs.
FlavourTag-Collection	The name of the collection of the flavour tag results that will be created.
Filename-b_net-1vtx	Filename for the 1 vertex b tag network.
Filename-c_net-1vtx	Filename for the 1 vertex c tag network.
Filename-bc_-net-1vtx	Filename for the 1 vertex c tag (only b background) network.
Filename-b_net-2vtx	Filename for the 2 vertex b tag network.
Filename-c_net-2vtx	Filename for the 2 vertex c tag network.
Filename-bc_-net-2vtx	Filename for the 2 vertex c (only b background) tag network.
Filename-b_net-3plusvtx	Filename for the 3 or more vertices b tag network.
Filename-c_net-3plusvtx	Filename for the 3 or more vertices c tag network.
Filename-bc_-net-3plusvtx	Filename for the 3 or more vertices c tag (only b background) network.

Author

Mark Grimes (mark.grimes@bristol.ac.uk)

The documentation for this class was generated from the following file:

- FlavourTag.h

12.9 vertex_lcfi::ZVTOP::GaussEllipsoid Class Reference

Gaussian ellipsoid component of the vertex function.

`#include <gaussellipsoid.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::GaussEllipsoid:

Collaboration diagram for vertex_lcfi::ZVTOP::GaussEllipsoid:

Public Member Functions

- GaussEllipsoid (InteractionPoint *IP)
  Construct from an interaction point.
- double valueAt (const Vector3 &Point) const
  Calculate the value of the ellipsoid at a point.
- InteractionPoint * ip ()
  InteractionPoint object used.

12.9.1 Detailed Description

Gaussian ellipsoid component of the vertex function.

Gaussian Ellipsoid in detector space (usually representing IP) The ellipsoid position and size are determined by the position and covariance matrix of a given InteractionPoint by:

$$V(\mathbf{r}) = e^{-0.5rV^{-1}r^T}$$

where r is the vector from ip position to query point and V is the covariance matrix. (Both in x,y,z space)

Note this is a deliberatly unnormalised gaussian.

The interaction point is not modified at any point by this class

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

12.9.2 Constructor & Destructor Documentation

12.9.2.1 vertex_lcfi::ZVTOP::GaussEllipsoid::GaussEllipsoid (InteractionPoint * *IP*)

Construct from an interaction point.

Parameters

IP	Pointer to InteractionPoint to use

12.9.3 Member Function Documentation

12.9.3.1 InteractionPoint* vertex_lcfi::ZVTOP::GaussEllipsoid::ip ( )

InteractionPoint object used.

Returns

Pointer to InteractionPoint used by this instance

12.9.3.2 double vertex_lcfi::ZVTOP::GaussEllipsoid::valueAt (const Vector3 & *Point*) const [virtual]

Calculate the value of the ellipsoid at a point.

Parameters

Point	Vector3 of the spatial point

Returns

Value of ellipsoid at point.

Implements vertex_lcfi::ZVTOP::VertexFunctionElement.

The documentation for this class was generated from the following file:

- gaussellipsoid.h

12.10 vertex_lcfi::ZVTOP::GaussTube Class Reference

Gaussian tube component of the vertex function.

`#include <gausstube.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::GaussTube:

Collaboration diagram for vertex_lcfi::ZVTOP::GaussTube:

Public Member Functions

- GaussTube (Track *Track)
  Construct from a Track, makes a trackstate for its own use.
- ~GaussTube ()
  Delete tube and trackstate used.
- double valueAt (const Vector3 &Point) const
  Calculate the value of the tube at point.

12.10.1 Detailed Description

Gaussian tube component of the vertex function.

Gaussian Tube in detector space (usually representing a track) The track position and size are determined by the position and covariance matrix of a given Track by:

$$V(\mathbf{r}) = e^{-0.5rV^{-1}r^T}$$

where r is the vector from to query point to the closest point on the track and V is the covariance matrix of the track. (Both in rPhi,z space)

Note this is a deliberatly unnormalised gaussian.

The Track is not modified at any point by this class

This guassian tube makes its own trackstate object from the track given at construction which it uses to perform the calculation.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

12.10.2 Constructor & Destructor Documentation

12.10.2.1 vertex_lcfi::ZVTOP::GaussTube::GaussTube (Track * *Track*)

Construct from a Track, makes a trackstate for its own use.

As a trackstate is made from the track, note that any changes to the track will not propogate to the tube.

Parameters

Track	Track that forms the guassian tube

12.10.3 Member Function Documentation

12.10.3.1 double vertex_lcfi::ZVTOP::GaussTube::valueAt (const Vector3 & *Point*) const [virtual]

Calculate the value of the tube at point.

Swims the track to the point of closest approach and calculate the tube value.

Parameters

Point	Vector3 of the spacial point

Returns

Value of tube at point

Implements vertex_lcfi::ZVTOP::VertexFunctionElement.

The documentation for this class was generated from the following file:

- gausstube.h

12.11 vertex_lcfi::ZVTOP::GhostFinderStage1 Class Reference

First stage of ghost track alorithm - ghost finder.

`#include <ghostfinderstage1.h>`

Public Member Functions

- GhostFinderStage1 ()
  Default Constructor.
- Track * findGhost (double InitialWidth, double MaxChi2Allowed, const Vector3 &JetDir, const std::vector< Track * > &JetTracks, InteractionPoint *IP)
  Find the ghost track.

12.11.1 Detailed Description

First stage of ghost track alorithm - ghost finder.

From a given set of Tracks and an InteractionPoint find a track with a direction and width consistant with a decaying particle forming the jet.

Note that currently the ghost track always originates at the origin and ignores the position of the Interaction Point. This should be fixed in a future release.

Todo Upgrade to movable IP

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

12/12/06

12.11.2 Constructor & Destructor Documentation

12.11.2.1 vertex_lcfi::ZVTOP::GhostFinderStage1::GhostFinderStage1 ( )

Default Constructor.

Creates a finder

12.11.3 Member Function Documentation

12.11.3.1 Track* vertex_lcfi::ZVTOP::GhostFinderStage1::findGhost (double *InitialWidth,* double *MaxChi2Allowed,* const Vector3 & *JetDir,* const std::vector< Track * > & *JetTracks,* InteractionPoint * *IP*)

Find the ghost track.

Using a given initial width and direction the ghost track is swivelled in Phi and Theta to minimise the chi sqaured to the other tracks. The ghost track width is then inflated to be consistant with the set of tracks to the level of MaxChiAllowed. The minimisation is then repeated and the width adjusted again.

There are some details of the minimisation which are deatiled in the GhostTrack paper

The documentation for this class was generated from the following file:

- ghostfinderstage1.h

12.12 vertex_lcfi::util::HelixRep Class Reference

Multi-Perpose 3 Vector.

`#include <helixrep.h>`

12.12.1 Detailed Description

Multi-Perpose 3 Vector.

5 Param Helix Representation

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

13/02/06

The documentation for this class was generated from the following file:

- helixrep.h

12.13 vertex_lcfi::ZVTOP::InteractionPoint Class Reference

Interaction Point representation.

`#include <interactionpoint.h>`

Public Member Functions

- InteractionPoint (const Vector3 &Position, const SymMatrix3x3 &ErrorMatrix)
  Construct from a point and error matrix.
- double distanceTo (const Vector3 &Point) const
  Return distance from this interacion point to a point.
- const Vector3 & position () const
  Return position of IP.
- const SymMatrix3x3 & errorMatrix () const
  Return error of IP.
- const Matrix3x3 & inverseErrorMatrix () const
  Return inverse error of IP.
- double chi2 (const Vector3 &Point) const
  Return chi squared of IP to a point.

12.13.1 Detailed Description

Interaction Point representation.

Provides ZVTOP with information about the IP if known. Used in the VertexFunction, and may be used in a VertexFitter

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

12.13.2 Constructor & Destructor Documentation

12.13.2.1 vertex_lcfi::ZVTOP::InteractionPoint::InteractionPoint (const Vector3 & *Position,* const SymMatrix3x3 & *ErrorMatrix*)

Construct from a point and error matrix.

Parameters

Position	Vector3 of the IP's location
ErrorMatrix	Matrix3x3 of the IP's error

12.13.3 Member Function Documentation

12.13.3.1 double vertex_lcfi::ZVTOP::InteractionPoint::chi2 (const Vector3 & *Point*) const

Return chi squared of IP to a point.

Returns

double of chi squared

12.13.3.2 double vertex_lcfi::ZVTOP::InteractionPoint::distanceTo (const Vector3 & *Point*) const

Return distance from this interacion point to a point.

Parameters

Point	Vector3 of point to measure distance to

Returns

distance between ip and point

12.13.3.3 const SymMatrix3x3& vertex_lcfi::ZVTOP::InteractionPoint::errorMatrix ( ) const

Return error of IP.

Returns

SymMatrix3x3 of IP's error

12.13.3.4 const Matrix3x3& vertex_lcfi::ZVTOP::InteractionPoint::inverseErrorMatrix ( ) const

Return inverse error of IP.

Returns

Matrix3x3 of IP's inverse error

12.13.3.5 const Vector3& vertex_lcfi::ZVTOP::InteractionPoint::position ( ) const

Return position of IP.

Returns

Vector3 of IP's position

The documentation for this class was generated from the following file:

- interactionpoint.h

12.14 vertex_lcfi::Jet Class Reference

Simple Jet Class.

`#include <jet.h>`

Public Member Functions

- Jet ()
  Default Constructor.
- Jet (Event *Event, const std::vector< Track * > &Tracks, double _Energy, Vector3 Momentum, void *TrackingNum)
  Full Constructor.
- Event * event () const
  Event.
- const std::vector< Track * > & tracks () const
  Tracks.
- void * trackingNum () const
  Tracking Number.
- void addTrack (Track *Track)
  Add Track.
- bool removeTrack (Track *TrackR)
  Remove Track.
- bool hasTrack (Track *Track) const
  Does the Jet have this Track?
- Vector3 momentum () const
  Momentum.
- double energy () const
  Energy.

12.14.1 Detailed Description

Simple Jet Class.

Description

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

12.14.2 Constructor & Destructor Documentation

12.14.2.1 vertex_lcfi::Jet::Jet (Event * *Event*, const std::vector< Track * > & *Tracks*, double *_Energy*, Vector3 *Momentum*, void * *TrackingNum*)

Full Constructor.

Parameters

Event	Pointer to the event containing this jet
Tracks	Vector of pointers to the tracks in this jet

Returns

Return

12.14.3 Member Function Documentation

12.14.3.1 void vertex_lcfi::Jet::addTrack (Track * *Track*)

Add Track.

Add a track to the jet

Parameters

Track	Pointer to the track to be added

12.14.3.2 double vertex_lcfi::Jet::energy ( ) const [inline]

Energy.

Sum energy of the jet

Returns

double of the energy

12.14.3.3 Event* vertex_lcfi::Jet::event ( ) const

Event.

Returns

A pointer to this jets event

12.14.3.4 bool vertex_lcfi::Jet::hasTrack (Track * *Track*) const

Does the Jet have this Track?

Does this jet have a perticular track?

Parameters

Track	Pointer to track to be checked

Returns

1 if track was found, 0 if not found

12.14.3.5 Vector3 vertex_lcfi::Jet::momentum ( ) const [inline]

Momentum.

The average perigee momentum of the tracks in the jet

Returns

Vector3 of the momentum

12.14.3.6 bool vertex_lcfi::Jet::removeTrack (Track * *TrackR*)

Remove Track.

Remove a track from the jet

Parameters

TrackR	Pointer to track to be removed

Returns

1 if track was found and removed, 0 if not found

12.14.3.7 void* vertex_lcfi::Jet::trackingNum ( ) const

Tracking Number.

Returns

Integer number

12.14.3.8 const std::vector<Track*>& vertex_lcfi::Jet::tracks ( ) const

Tracks.

Returns

A Vector of pointers to the tracks in the jet

The documentation for this class was generated from the following file:

- jet.h

12.15 vertex_lcfi::JointProb Class Reference

Calculation of the Joint probability flavour tag inputs.

`#include <jointprob.h>`

Inheritance diagram for vertex_lcfi::JointProb:

Collaboration diagram for vertex_lcfi::JointProb:

Public Member Functions

- JointProb ()
  Constructor.
- string name () const
  Name.
- std::vector< string > parameterNames () const
  Parameter Names.
- std::vector< string > parameterValues () const
  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)
  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)
  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)
  Set Pointer Parameter.
- std::map< Projection, double > calculateFor (Jet *MyJet) const
  Run the algorithm on a jet.

12.15.1 Detailed Description

Calculation of the Joint probability flavour tag inputs.

This class calculates the joint probability flavour tag input. This is the probability that all the tracks of a Jet come from the primary vertex. The calculation is done by using the information on the impact parameters of each Track. The class outputs three values : the calculation of joint probability in the RPhi plane, in the Z direction and in 3 dimensions. Please note that while the first two have been extensively tested the third has not since it is not used as a flavour tagging inputs parameter.

Parameters

MaxD0-Significance	Maximum d0 significance. All more significant Tracks get cut.
MaxD0andZ0	Maximum value of d0 and of z0.
Resolution-ParameterRphi	This is a Pointer parameter. It must point to a vector containing 5 double elements. These are the standard deviations of the impact parameter significances of the distribution of tracks coming from the primary vertex. These values can be obtained from a fit to the impact parameter distribution of tracks behind the interaction point, i.e. with negative impact parameter, where the direction of reference is the Jet direction. Please note that ideally these parameters should be calculated whenever the boundary conditions change in a way that affects the impact parameter distributions, by running a separate processor beforehand. This at the moment is not done and the values are inserted as parameters. The default values have been inherited from previous studies.
Resolution-ParameterZ	Similar as ResolutionParameterRphi, but for the Z direction.
Resolution-Parameter3D	Similar as before but for the 3-D calculation

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.15.2 Member Function Documentation

12.15.2.1 std::map<Projection, double> vertex_lcfi::JointProb::calculateFor (Jet * *MyJet*) const [virtual]

Run the algorithm on a jet.

Calculate the jet joint probability flavour tagging parameter for the jet and output the joint probability for RPhi, Z and ThreeD

Parameters

Jet	Pointer to jet to be analysed

Returns

Map containing the following keys: RPhi, Z and ThreeD

Implements vertex_lcfi::Algo< Jet *, std::map< Projection, double > >.

12.15.2.2 string vertex_lcfi::JointProb::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< Jet *, std::map< Projection, double > >.

12.15.2.3 std::vector<string> vertex_lcfi::JointProb::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< Jet *, std::map< Projection, double > >.

12.15.2.4 std::vector<string> vertex_lcfi::JointProb::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< Jet *, std::map< Projection, double > >.

12.15.2.5 void vertex_lcfi::JointProb::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.15.2.6 void vertex_lcfi::JointProb::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.15.2.7 void vertex_lcfi::JointProb::setStringParameter (const string & *Parameter*, const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- jointprob.h

12.16 LCFIAIDAPlotProcessor Class Reference

LCFIAIDAPlotProcessor Class - make plots of the LCFI flavour tag and vertex charge code.

`#include <LCFIAIDAPlotProcessor.h>`

Inherits Processor.

Protected Member Functions

- int FindTrueJetType (LCEvent *pEvent, unsigned int jetNumber)

  Finds the true flavour of a jet (uses TrueJetFlavourCollection)
- float FindTrueJetHadronCharge (LCEvent *pEvent, unsigned int jetNumber)

  Finds the true charge of the hadron producing a jet (uses TrueJetFlavourCollection)
- int FindTrueJetPDGCode (LCEvent *pEvent, unsigned int jetNumber)

  Finds the PDG code of the hadron producing a jet (uses TrueJetFlavourCollection)
- float FindTrueJetPartonCharge (LCEvent *pEvent, unsigned int jetNumber)

  Finds the true charge of the parton producing a jet (uses TrueJetFlavourCollection)
- int FindTrueJetFlavour (LCEvent *pEvent, unsigned int jetNumber)

  Finds the true flavour of the jet (uses TrueJetFlavourCollection)
- void FindTrueJetDecayLength (LCEvent *pEvent, unsigned int jetNumber, std::vector< double > &decaylengthvector, std::vector< double > &bjetdecaylengthvector, std::vector< double > &cjetdecaylengthvector)

  Finds the true decay length of the longest b- or c- hadron in a jet.
- int FindNumVertex (LCEvent *pEvent, unsigned int jetNumber, unsigned int iInputsCollection)

  Finds the number of vertices in an event (from the flavour tag inputs)
- int FindCQVtx (LCEvent *pEvent, unsigned int jetNumber)

  Finds the vertex charge of the jet - using cuts tuned to find vertex charge for C-jets (from CVertexChargeCollection)
- int FindBQVtx (LCEvent *pEvent, unsigned int jetNumber)

  Finds the vertex charge of the jet - using cuts tuned to find vertex charge for B-jets (from BVertexChargeCollection)
- AIDA::IDataPointSet * CreateEfficiencyPlot (const AIDA::IHistogram1D *pSignal, AIDA::IDataPointSet *pDataPointSet)

  Makes a DataPointSet of the tag efficiency e.g number of B-jets passing a given B-tag NN cut, as a function of NN.
- AIDA::IDataPointSet * CreateEfficiencyPlot2 (const AIDA::IHistogram1D *pAllEvents, const AIDA::IHistogram1D *pPassEvents, AIDA::IDataPointSet *pDataPointSet)

  Makes a DataPointSet of histogram 1 divide by histogram 2 - this is an IDataPointSet as a histrogram gives the wrong errors.
- AIDA::IDataPointSet * CreateIntegralPlot (const AIDA::IHistogram1D *pNN, AIDA::IDataPointSet *pIntegral)

  Makes a DataPointSet integrating a histogram from the first bin to the last bin – NOT USED.
- AIDA::IDataPointSet * CreatePurityPlot (const AIDA::IHistogram1D *pSignal, const AIDA::IHistogram1D *pBackground, AIDA::IDataPointSet *pDataPointSet)

  Makes a DataPointSet of the tag purity e.g. N(B-jets passing NN cut)/N(all-jets passing NN cut) for a given B-tag NN cut, as a function of NN.
- AIDA::IDataPointSet * CreateLeakageRatePlot (const AIDA::IHistogram1D *pBackground, AIDA::IDataPointSet *pDataPointSet)

  Makes a DataPointSet showing the tagging leakage e.g. the number of non-B-jets passing a given B-tag NN cut, as a function of NN.
- AIDA::IDataPointSet * CreateXYPlot (const AIDA::IDataPointSet *pDataPointSet0, const AIDA::IDataPointSet *pDataPointSet1, AIDA::IDataPointSet *xyPointSet, const int dim0=0, const int dim1=0)

  Plots two DataPointSets against each other.
- AIDA::IHistogram1D * CreateIntegralHistogram (const AIDA::IHistogram1D *pNN, AIDA::IHistogram1D *pIntegral)

  Makes a histogram integrating a histogram from the first bin to the last bin - THE ERRORS RETURNED ARE WRONG!
- void CreateVertexChargeLeakagePlot (AIDA::IDataPointSet *pBJetVtxChargeDPS, AIDA::IDataPointSet *pCJetVtxChargeDPS)

  Makes DataPointSets for the number of.

Protected Attributes

- std::vector< std::string > _FlavourTagCollectionNames

  required input collections
- double _CosThetaJetMax

  cuts on all jets
- double _CosThetaJetMin

  cuts on all jets
- double _PJetMin

  cuts on all jets
- double _PJetMax

  cuts on all jets
- double _BTagNNCut

  Cut on the NN output variables - applied in vertex charge plots.
- double _CTagNNCut

  Cut on the NN output variables - applied in vertex charge plots.
- bool _PrintTrackVertexOutput

  optional parameters to make an ntuple of the neural net inputs; and print out the tagging ouputs (useful for scripts)
- AIDA::IHistogram2D * _pBJetCharge2D

  True B-jets - vertex charge vs true charge.
- AIDA::IHistogram2D * _pCJetCharge2D

  True C-jets - vertex charge vs true charge.
- AIDA::IHistogram1D * _pBJetLeakageRate

  True B-jets - vertex charge leakage rate.
- AIDA::IHistogram1D * _pCJetLeakageRate

  True C-jets - vertex charge leakage rate.
- AIDA::IHistogram1D * _pBJetVertexCharge

  True B-jets - vertex charge.
- AIDA::IHistogram1D * _pCJetVertexCharge

  True C-jets - vertex charge.
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _inputsHistogramsBJets

  Histograms of the neural net inputs for true B-jets.
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _inputsHistogramsCJets

  Histograms of the neural net inputs for true C-jets.
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _inputsHistogramsUDSJets

  Histograms of the neural net inputs for light B-jets.
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _zoomedInputsHistogramsBJets

  Zoomed-in histograms of some of the neural net inputs for true B-jets.
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _zoomedInputsHistogramsCJets

  Zoomed-in histograms of some of the neural net inputs for true C-jets.
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _zoomedInputsHistogramsUDSJets

Zoomed-in histograms of some of the neural net inputs for true light-jets.

- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pLightJetBTag

  Histograms of the neural net B-tag outputs for true light-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pLightJetCTag

  Histograms of the neural net C-tag outputs for true light-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pBJetBTag

  Histograms of the neural net B-tag outputs for true B-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pBJetCTag

  Histograms of the neural net C-tag outputs for true B-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pCJetBTag

  Histograms of the neural net B-tag outputs for true C-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pCJetCTag

  Histograms of the neural net C-tag outputs for true C-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pBJetBCTag

  Histograms of the neural net BC-tag outputs for true B-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pCJetBCTag

  Histograms of the neural net BC-tag outputs for true C-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pLightJetBCTag

  Histograms of the neural net BC-tag outputs for true light-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pBTagBackgroundValues

  Histograms of the neural net B-tag outputs for non B-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pCTagBackgroundValues

  Histograms of the neural net C-tag outputs for non C-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)

- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pBCTagBackgroundValues

  Histograms of the neural net BC-tag outputs for non C-jets - seperately for different number of vertices in the jets, 1, 2, >=3, any (sum of previous)
- std::vector< std::map < std::string, AIDA::IHistogram1D * > > _pBJetBTagIntegral

  Histograms of the neural net tags - number of events that pass a given cut: jet NN value > given NN value for the three tags - B-tag, C-tag, BC-tag.
- int _numberOfPoints

  Number of bins used for neural nets plots.
- AIDA::ITuple * _pMyTuple

  Tuple of the input variables - only filled for one input collection - selected with UseFlavourTagCollectionForVertexCharge.
- int _cJet_truePlus2

  numbers of true C-jets with true charge ++
- int _cJet_truePlus

  numbers of true C-jets with true charge +
- int _cJet_trueNeut

  numbers of true C-jets with true charge 0
- int _cJet_trueMinus

  numbers of true C-jets with true charge -
- int _cJet_trueMinus2

  numbers of true C-jets with true charge –
- int _cJet_truePlus2_recoPlus

  numbers of true C-jets with true charge ++; reconstructed vertex charge >0
- int _cJet_truePlus2_recoNeut

  numbers of true C-jets with true charge ++; reconstructed vertex charge =0
- int _cJet_truePlus2_recoMinus

  numbers of true C-jets with true charge ++; reconstructed vertex charge <0
- int _cJet_truePlus_recoPlus

  numbers of true C-jets with true charge +; reconstructed vertex charge >0
- int _cJet_truePlus_recoNeut

  numbers of true C-jets with true charge +; reconstructed vertex charge =0
- int _cJet_truePlus_recoMinus

  numbers of true C-jets with true charge +; reconstructed vertex charge <0
- int _cJet_trueNeut_recoPlus

  numbers of true C-jets with true charge 0; reconstructed vertex charge >0
- int _cJet_trueNeut_recoNeut

  numbers of true C-jets with true charge 0; reconstructed vertex charge =0
- int _cJet_trueNeut_recoMinus

  numbers of true C-jets with true charge 0; reconstructed vertex charge <0
- int _cJet_trueMinus_recoPlus

  numbers of true C-jets with true charge -; reconstructed vertex charge >0
- int _cJet_trueMinus_recoNeut

  numbers of true C-jets with true charge -; reconstructed vertex charge =0
- int _cJet_trueMinus_recoMinus

  numbers of true C-jets with true charge -; reconstructed vertex charge <0
- int _cJet_trueMinus2_recoPlus

  numbers of true C-jets with true charge –; reconstructed vertex charge >0
- int _cJet_trueMinus2_recoNeut

numbers of true C-jets with true charge –; reconstructed vertex charge =0

- int _cJet_trueMinus2_recoMinus

  numbers of true C-jets with true charge –; reconstructed vertex charge <0

- int _bJet_truePlus2

  numbers of true B-jets with true charge ++

- int _bJet_truePlus

  numbers of true B-jets with true charge +

- int _bJet_trueNeut

  numbers of true B-jets with true charge 0

- int _bJet_trueMinus

  numbers of true B-jets with true charge -

- int _bJet_trueMinus2

  numbers of true B-jets with true charge –

- int _bJet_truePlus2_recoPlus

  numbers of true B-jets with true charge ++; reconstructed vertex charge >0

- int _bJet_truePlus2_recoNeut

  numbers of true B-jets with true charge ++; reconstructed vertex charge =0

- int _bJet_truePlus2_recoMinus

  numbers of true B-jets with true charge ++; reconstructed vertex charge <0

- int _bJet_truePlus_recoPlus

  numbers of true B-jets with true charge +; reconstructed vertex charge >0

- int _bJet_truePlus_recoNeut

  numbers of true B-jets with true charge +; reconstructed vertex charge =0

- int _bJet_truePlus_recoMinus

  numbers of true B-jets with true charge +; reconstructed vertex charge <0

- int _bJet_trueNeut_recoPlus

  numbers of true B-jets with true charge 0; reconstructed vertex charge >0

- int _bJet_trueNeut_recoNeut

  numbers of true B-jets with true charge 0; reconstructed vertex charge =0

- int _bJet_trueNeut_recoMinus

  numbers of true B-jets with true charge 0; reconstructed vertex charge <0

- int _bJet_trueMinus_recoPlus

  numbers of true B-jets with true charge -; reconstructed vertex charge >0

- int _bJet_trueMinus_recoNeut

  numbers of true B-jets with true charge -; reconstructed vertex charge =0

- int _bJet_trueMinus_recoMinus

  numbers of true B-jets with true charge -; reconstructed vertex charge <0

- int _bJet_trueMinus2_recoPlus

  numbers of true B-jets with true charge –; reconstructed vertex charge >0

- int _bJet_trueMinus2_recoNeut

  numbers of true B-jets with true charge –; reconstructed vertex charge =0

- int _bJet_trueMinus2_recoMinus

  numbers of true B-jets with true charge –; reconstructed vertex charge <0

- std::vector< unsigned int > _cJet_truePlus2_angle

  Vector of numbers of true C-jets with true charge ++ See above for details.

- int _nb_twoVertex_bTrack_Primary

  Numbers for purity if reconstructed track-vertex association.

Static Protected Attributes

- static const unsigned int N_VERTEX_CATEGORIES =3

  number of different vertex categories we want to look at: 1 vertex, 2 vertices, >=3 vertices
- static const int N_JETANGLE_BINS =10

  number of bins used in vertex charge leakage plots

12.16.1 Detailed Description

LCFIAIDAPlotProcessor Class - make plots of the LCFI flavour tag and vertex charge code.

Please note that LCFIAIDAPlotProcessor will not compile with RAIDA v01-03 To use LCFIAIDAPlotProcessor please use AIDAJNI for the implementation of AIDA run "cmake -DBUILD_WITH="ROOT;AIDAJNI" -DAIDAJNI_HOME=${AIDAJNI_HOME}"

This sorry states of affairs comes about becayse not all AIDA functions are defined in RAIDA v01-03

In order to make a histogram file, LCFIAIDAPlotProcessor must be run with AIDAProcessor.

LCFIAIDAPlotProcessor reads in one (or more) FlavourTagCollections, e.g. from FlavourTag and one (or more) TagInputCollections. Histograms/plots are made of the neural net outputs, the purity and leakage rate of the flavour tag. These are split into sub-samples based on the number of vertices found in the jets. Plots are also made of the inputs to the FlavourTagCollections - split into sub-samples based on the true (MC) flavour of the jet.

Options are given to make a tuple of the flavour tag inputs and to print out a text file of the different flavour tag neural net outputs.

(When providing more than one FlavourTagCollection and/or TagInputCollection plots for each collection will be made in different directories.)

In addition LCFIAIDAPlotProcessor also requires a jet collection, and the following collections, which should refer to the *same* jet collection.

BVertexChargeCollection – calculated in VertexChargeProcessor

CVertexChargeCollection – calculated in VertexChargeProcessor

TrueJetFlavourCollection – calculated in TrueAngularJetFlavourProcessor

Input

The following collections must be available:

Parameters

FlavourTagCollections	StringVec of LCFloatVec names representing the flavour tag inputs collections - may be more than one collection.
TagInputsCollections	StringVec of LCFloatVec names the flavour tag input collections - may be more than one collection.
JetCollectionName	Name of ReconstructedParticleCollection representing the jets.
VertexCollectionName	Name of VertexCollection representing the Vertex collection of the jets.
BVertexChargeCollection	Name of LCFloatVector of the vertex charge of the jet collection, assuming the jets are b-jets (calculated in VertexChargeProcessor)
CVertexChargeCollection	Name of LCFloatVector of the vertex charge of the jet collection, assuming the jets are c-jets (calculated in VertexChargeProcessor)
TrueJetFlavourCollection	Name of LCFloatVector of the true (MC) flavour of the jets (calculated in TrueAngularJetFlavourProcessor)

VertexCollection-Name	Name of VertexCollection representing the vertices.
BTagNNCut	Double reprsenting the lower cut on the b-tag NN value for some of the plots.
CTagNNCut	Double reprsenting the lower cut on the c-tag NN value for some of the plots.
CosThetaJetMax	Double representing upper cut on cos(theta) of the jets for the plots.
CosThetaJetMin	Double representing lower cut on cos(theta) of the jets for the plots.
PJetMax	Double representing upper cut on momentum of the jet for the plots.
PJetMin	Double representing lower cut on momentum of the jet for the plots.
MakeTuple	Bool set true if you want to make a tuple of the TagInputCollection variables.
NeuralNet-OutputFile	String representing name of text file of neural net values to. Only used if PrintNeuralNetOutput parameter is true. If left blank, output will be directed to standard out
PrintNeuralNet-Output	Bool set true if you want to make a text file of the neural net values (useful for some scripts).
UseFlavourTag-CollectionFor-VertexCharge	For vertex charge plots we demand the cTag>CTagNNCut and bTag>BTagNNCut. This integer is used if there is more than one tag collection, to determine which of the collections should be used to apply this cut.

Output

- An aida (or root??) file containing the histograms, plots and tuples.
- (Optionally) a text file containing some of the neural net tagging output

Author

Victoria Martin (victoria.martin@ed.ac.uk)

12.16.2 Member Data Documentation

12.16.2.1 std::vector< std::map<std::string,AIDA::IHistogram1D*> > LCFIAIDAPlotProcessor::_pBJetBTagIntegral [protected]

Histograms of the neural net tags - number of events that pass a given cut: jet NN value > given NN value for the three tags - B-tag, C-tag, BC-tag.

- separately for true B jets, true C jets & true light jets and different number of vertices in the jets, 1, 2 or >=3 & any (sum of previous three) See comments above

The documentation for this class was generated from the following file:

- LCFIAIDAPlotProcessor.h

12.17 vertex_lcfi::MemoryManager< T > Class Template Reference

Memory management.

`#include <memorymanager.h>`

Inheritance diagram for vertex_lcfi::MemoryManager< T >:

Collaboration diagram for vertex_lcfi::MemoryManager< T >:

Public Member Functions

- virtual ~MemoryManager ()

  Destructor - will delete all held objects.

- void registerObject (T *pointer)

  Register an object for memory management.
- void delAll ()

  Delete all objects held by this MemoryManager.

Static Public Member Functions

- static MemoryManager< T > * Event ()

  Returns the Event duration singleton instance of the MemoryManager for type T.
- static MemoryManager< T > * Run ()

  Returns the Run duration singleton instance of the MemoryManager for type T.

Protected Member Functions

- MemoryManager ()

  Do not use.
- MemoryManager (const MemoryManager< T > &)

  Do not use.
- MemoryManager< T > & operator= (const MemoryManager< T > &)

  Do not use.

12.17.1 Detailed Description

template<class T>class vertex_lcfi::MemoryManager< T >

Memory management.

Memory in the vertex package is handled on a run and event basis If you wish to have an object that lasts the length of the event then call:

```
myType* myObject = new myType(construction parameters);

MemoryManager<myType>::Event()->registerObject(myObject);
```

At the end of the event to free all objects of all types made using the above call:

```
MetaMemoryManager::Event()->delAllObjects();
```

Similarly for run lifetime objects, replacing Event with Run.

The documentation for this class was generated from the following file:

- memorymanager.h

12.18 vertex_lcfi::MemoryManagerType Class Reference

Base class for all MemoryManagers.

`#include <memorymanager.h>`

Inheritance diagram for vertex_lcfi::MemoryManagerType:

Public Member Functions

- virtual ~MemoryManagerType ()

  Empty Destructor.
- virtual void delAll ()=0

  Delete all objects that this MemoryManger has pointers to.

12.18.1 Detailed Description

Base class for all MemoryManagers.

The documentation for this class was generated from the following file:

- memorymanager.h

12.19 vertex_lcfi::MetaMemoryManager Class Reference

MemoryManager Controller - see MemoryManager.

`#include <memorymanager.h>`

Public Member Functions

- void delAllObjects ()

  Delete all objects of all types held by this instance.
- void registerType (MemoryManagerType *Type)

  Used by the MemoryManager of each type to alert the controller of its existance.

Static Public Member Functions

- static MetaMemoryManager * Event ()

  Returns the Event duration singleton instance of the controller.
- static MetaMemoryManager * Run ()

  Returns the Run duration singleton instance of the controller.

Protected Member Functions

- MetaMemoryManager ()

  Do not use.
- MetaMemoryManager (const MetaMemoryManager &)

  Do not use.
- MetaMemoryManager & operator= (const MetaMemoryManager &)

  Do not use.

12.19.1 Detailed Description

MemoryManager Controller - see MemoryManager.

Keeps track of MemoryManagers for each type, and tells them to delete their objects when delAllObjects is called.

The documentation for this class was generated from the following file:

- memorymanager.h

12.20 NeuralNetTrainerProcessor Class Reference

Trains neural networks to be used for jet flavour tagging.

`#include <NeuralNetTrainer.h>`

Inherits Processor.

12.20.1 Detailed Description

Trains neural networks to be used for jet flavour tagging.

Trains flavour tagging networks using the BackPropagationCGAlgorithm (see the Neural Net Package page) with 500 epochs. The networks are trained on the following data:

If only 1 vertex is found (i.e. only the interaction point)

$$tanh\left(\frac{D0Significance1}{100}\right)$$

$$tanh\left(\frac{D0Significance2}{100}\right)$$

$$tanh\left(\frac{Z0Significance1}{100}\right)$$

$$tanh\left(\frac{Z0Significance2}{100}\right)$$

$$JointProbRPhi$$

$$JointProbZ$$

$$tanh\left(\frac{3\times Momentum1}{E}\right)$$

$$tanh\left(\frac{3\times Momentum2}{E}\right)$$

If 2 or more vertices are found

$$tanh\left(\frac{DecayLengthSignificance}{6\times E}\right)$$

$$tanh\left(\frac{DecayLength}{10}\right)$$

$$tanh\left(\frac{PTCorrectedMass}{5}\right)$$

$$tanh\left(\frac{RawMomentum}{E}\right)$$

$$JointProbRPhi$$

$$JointProbZ$$

$$tanh\left(\frac{NumTracksInVertices}{10}\right)$$

$$SecondaryVertexProbability$$

Where E is the jet energy, everything else is the data calculated by FlavourTagInputsProcessor.

Note that **the processor applies it's own hard coded cuts**. These are documented under the full description of NeuralNetTrainerProcessor::_passesCuts().

Input

- A collection of ReconstructedParticles that represents the jets in the event (put in by your jet finder, say SatoruJetFinderProcessor).
- A LCFloatVec collection that holds the true jet flavours, in the same order as the jets (put in by TrueAngularJetFlavourProcessor).
- A LCFloatVec collection that holds the flavour tag inputs (put in by FlavourTagInputsProcessor)
- Between 1 and 9 filenames for the generated networks. If an output filename is left blank then that network is not trained, but if none are supplied then a lcio::Exception is thrown.

Output

Trained neural networks to the filenames supplied, in the format requested. The LCIO file is not modified at all.

Parameters

JetCollectionName	Name of the ReconstructedParticle collection that represents jets.
FlavourTagInputsCollection	Name of the LCFloatVec collection that holds the flavour tag inputs.
TrueJetFlavourCollection	Name of the LCIntVec Collection that contains the true jet flavours.
Filename-b_net-1vtx	Output filename for the trained 1 vertex b-tag net.
Filename-c_net-1vtx	Output filename for the trained 1 vertex c-tag net.
Filename-bc_net-1vtx	Output filename for the trained 1 vertex c-tag (with only b background) net.
Filename-b_net-2vtx	Output filename for the trained 2 vertex b-tag net.
Filename-c_net-2vtx	Output filename for the trained 2 vertex c-tag net.
Filename-bc_net-2vtx	Output filename for the trained 2 vertex c-tag (with only b background) net.
Filename-b_net-3plusvtx	Output filename for the trained 3 or more vertices b-tag net.
Filename-c_net-3plusvtx	Output filename for the trained 3 or more vertices c-tag net.
Filename-bc_net-3plusvtx	Output filename for the trained 3 or more vertices c-tag (with only b background) net.

Author

Mark Grimes (mark.grimes@bristol.ac.uk)

The documentation for this class was generated from the following file:

- NeuralNetTrainer.h

12.21 vertex_lcfi::ParameterSsignificance Class Reference

Calculation of a series of flavour tag inputs.

`#include <paramsignificance.h>`

Inheritance diagram for vertex_lcfi::ParameterSignificance:

Collaboration diagram for vertex_lcfi::ParameterSignificance:

Public Member Functions

- ParameterSignificance ()

  Constructor.
- string name () const

  Name.
- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- std::map< SignificanceType,
  double > calculateFor (Jet *MyJet) const

  Run the algorithm on a jet.

12.21.1 Detailed Description

Calculation of a series of flavour tag inputs.

This class finds the two most significant tracks in the RPhi plane. It then outputs the momentum, z significance and RPhi significance of these two tracks. The class has the following parameters:

Parameters

LayersHit	we will be setting minimum momentum cuts on tracks that will hit Layershit minus one and LayersHit or more layers of the vertex detector.
AllbutOne-Layers-MomentumCut	minimum momentum of particles that have been detected in LayersHit minus one layers of the vertex detector
AllLayers-MomentumCut	minimum momentum of particles that have been detected at least in LayersHit layers of the vertex detector
TwoTrackPidCut	this is a pointer parameter. This should point to the map outputted by the algorithm TwoTrackPid . This pointer is used to cut out the tracks that have been flagged as deriving from a gamma or a Ks.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.21.2 Member Function Documentation

12.21.2.1 std::map<SignificanceType, double> vertex_lcfi::ParameterSignificance::calculateFor (Jet * *MyJet*) const [virtual]

Run the algorithm on a jet.

Calculate the two tracks with the highest RPhi significance and output their RPhi significance, Z significance and momentum.

Parameters

Jet	Pointer to jet to be analysed

Returns

map containing the following keys:"D0SigTrack1", <D0SigTrack2","Z0SigTrack1", "Z0SigTrack2","Momentum-Track1","MomentumTrack2".

Implements vertex_lcfi::Algo< Jet *, std::map< SignificanceType, double > >.

12.21.2.2 string vertex_lcfi::ParameterSignificance::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< Jet *, std::map< SignificanceType, double > >.

12.21.2.3 std::vector<string> vertex_lcfi::ParameterSignificance::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< Jet *, std::map< SignificanceType, double > >.

12.21.2.4 std::vector<string> vertex_lcfi::ParameterSignificance::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< Jet *, std::map< SignificanceType, double > >.

12.21.2.5 void vertex_lcfi::ParameterSignificance::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.21.2.6 void vertex_lcfi::ParameterSignificance::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.21.2.7 void vertex_lcfi::ParameterSignificance::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- paramsignificance.h

12.22 vertex_lcfi::PerEventIPFitter Class Reference

Example jet variable that returns the number of tracks in the jet.

`#include <pereventipfitter.h>`

Inheritance diagram for vertex_lcfi::PerEventIPFitter:

Collaboration diagram for vertex_lcfi::PerEventIPFitter:

Public Member Functions

- string name () const
  Name.
- std::vector< string > parameterNames () const
  Parameter Names.
- std::vector< string > parameterValues () const
  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)
  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)
  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)
  Set Pointer Parameter.
- Vertex * calculateFor (Event *MyEvent) const
  Run the algorithm on an Event.

12.22.1 Detailed Description

Example jet variable that returns the number of tracks in the jet.

12.22.2 Member Function Documentation

12.22.2.1 Vertex* vertex_lcfi::PerEventIPFitter::calculateFor (Event * *MyEvent*) const [virtual]

Run the algorithm on an Event.

Calculate the DecayChain of the jet

Parameters

Event	Pointer to event to be analysed

Returns

Pointer to Vertex of algorithm result

Implements vertex_lcfi::Algo< Event *, Vertex * >.

12.22.2.2 string vertex_lcfi::PerEventIPFitter::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< Event *, Vertex * >.

12.22.2.3 std::vector<string> vertex_lcfi::PerEventIPFitter::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< Event *, Vertex * >.

12.22.2.4 std::vector<string> vertex_lcfi::PerEventIPFitter::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< Event *, Vertex * >.

12.22.2.5 void vertex_lcfi::PerEventIPFitter::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.22.2.6 void vertex_lcfi::PerEventIPFitter::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.22.2.7 void vertex_lcfi::PerEventIPFitter::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- pereventipfitter.h

12.23 PerEventIPFitterProcessor Class Reference

Determine IP position and error from the tracks in an event by simple fit.

`#include <PerEventIPFitter.h>`

Inherits Processor.

Collaboration diagram for PerEventIPFitterProcessor:

12.23.1 Detailed Description

Determine IP position and error from the tracks in an event by simple fit.

Inputs

Name	Type	Represents
InputRPCollectionName	ReconstructedParticle	Tracks to be fit

Outputs

Name	Type	Represents
VertexCollectionName	Vertex	Fitted IP

Description

This processor fits a set of LCIO ReconstructedParticles (which must have an LCIO Track attached to be used) to a common point. This is performed by iterative fitting, with removal of the track with highest chi-squared at each iteration until the fit reaches the probability threshold. If only one track remains then the default IP position and error are used. The result is stored as an LCIO Vertex.

This processor is highly unoptimised and untuned, and may take a long time to execute on a large set of tracks.

Currently uses VertexFitterLSM (from ZVTOP) to perform fitting.

Parameters

InputRP-Collection	Name of the ReconstructedParticle collection to be fit

VertexCollection-Name	Name of the output Vertex collection
DefaultIPPos	Length 3 Float Vector of position (x,y,z) returned (as LCIO Vertex) if no fit is found
DefaultIPErr	Length 6 Float Vector of covariance (lower symmetric) returned (as LCIO Vertex) if no fit is found
Probability-Threshold	Once the vertex is above this probability it is returned

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

The documentation for this class was generated from the following file:

- PerEventIPFitter.h

12.24 PlotProcessor Class Reference

Creates some sample plots from the data calculated by the LCFI vertex package.

`#include <PlotProcessor.h>`

Inherits Processor.

12.24.1 Detailed Description

Creates some sample plots from the data calculated by the LCFI vertex package.

An example of getting the flavour tag results from the LCIO file and plotting an efficiency purity graph with them. Also plots a graph of jet energies for good measure.

The processor checks the specifed LCFloatVec collections for the flavour tag values "BTag", "CTag" and "BCTag" which are the names that FlavourTagProcessor stores its b tag, c tag and c tag (only b background) values in respectively.

These values are checked against the true jet flavour (from the TrueJetFlavour LCIntVec) and efficiency-purity values calculated for a range of cuts.

The jet energy is taken from the energy of the reconstructed particle used to represent the jet.

Getting Root output

To output to a Root file instead of CSV files the processor has to be compiled with the USEROOT preprocessor flag defined. You could add "#define USEROOT" to the code, or more easily add the line

USERINCLUDES += -D USEROOT

to the userlib.gmk file that is in the Marlin directory. If Marlin is not already set up to use Root then you will also need to add the following lines (this assumes a fully working root installation):

USERINCLUDES += `root-config --cflags`

USERLIBS += `root-config --libs`

Input

From the LCIO file, flavour tag variable values of:

"BTag" "CTag" "BCTag"

And

"JetType"

Output

If the USEROOT preprocessor flag was defined when this processor was compiled, then the output will be a root file with the filename specified in the steering file. Otherwise, the efficiency-purity values will be output as comma separated values to the file <filename>+".csv", and the jet energies to <filename>+"-JetEnergies.csv".

Parameters

JetCollection-Name	Name of the ReconstructedParticle collection that represents jets.
FlavourTag-Collections	Names of the LCFloatVec collections holding the Flavour tags, all tags in this list will be produced in one file for comparison
TrueJetFlavour-Collection	LCIntVec that contains the MC Jet flavour (from TrueJetFlavourProcessor)
OutputFilename	The name of the file that will hold the output.

The documentation for this class was generated from the following file:

- PlotProcessor.h

12.25 RPCutProcessor Class Reference

Cuts ReconstuctedParticles(RPs) from a collection (or from a list of RPs held by another RP) based on several cut criteria.

`#include <RPCutProcessor.h>`

Inherits Processor.

12.25.1 Detailed Description

Cuts ReconstuctedParticles(RPs) from a collection (or from a list of RPs held by another RP) based on several cut criteria.

Input

Name	Type	Represents
InputRCPCollection	ReconstructedParticle	Collection to be cut

Output

Name	Type	Represents
OutputRCPCollection	ReconstructedParticle	If WriteNewCollection=true contains the RPs that passed the cuts.

Description

Based on several criteria this processor removes RPs from a collection, or if SubParticleLists = true then it removes RPs held by RPs in a collection.

Depending on WriteNewCollection, the Output is either the original collection with the RPs removed, or a new collection with the input collection remaining untouched.

NOTE - A track is cut if its ReconstructedParticle has no Track objects attached.

Most cuts follow a standard set of parameters:

a1_{CutName}Enable	If true the cut is enabled

a2_{CutName}CutLowerThan	If true RPs with a value lower than the cut value are cut, if false those higher than the cut value are cut.
a3_{CutName}CutValue	The value of the cut

(The letter and number index prefixed to each parameter are to ensure they stay together in the output of Marlin -x) The cuts that follow this scheme are:

Name	Description
Chi2OverDOF	Chi squared over degrees of freedom (Track::gethi2())
D0	Track D0 (Track::getD0())
D0Err	D0 Covariance (Track::getCovMatrix()[0])
Z0	Track Z0 (Track::getZ0())
Z0Err	Z0 Covariance (Track::getCovMatrix()[9])
PT	Track Pt (rPhi projection of Track::getMomentum())

Subdetector hits

The cut on subdetector hits relies on information in Track::getSubdetectorHitNumbers() this is an array. The processor is told what order the detectors are in this array by parameter "g2_SubDetectorNames" which is the sting names of the detectors in the same order. The other parameters then use these names.

MC PID of Parents

This cut uses MC information provided by the MCParticleRelation collection to cut particles whose parents have a PID in the list provided by parameter "h2_CutPIDS"

Bad parameters cut

If enabled by "i1_BadParametersEnable" this cut removes tracks with nan covariances and parameters.

MC Vertex cut

Experimental MC Cut - most likely removed in next release

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

The documentation for this class was generated from the following file:

- RPCutProcessor.h

12.26 vertex_lcfi::SecVertexProb Class Reference

Calculation of Secondary Vertex Probability.

`#include <secondvertexprob.h>`

Inheritance diagram for vertex_lcfi::SecVertexProb:

Collaboration diagram for vertex_lcfi::SecVertexProb:

Public Member Functions

- string name () const

  Name.
- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

Parameter Values.

- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- double calculateFor (DecayChain *MyDecayChain) const

  Run the algorithm on a Jet.

12.26.1 Detailed Description

Calculation of Secondary Vertex Probability.

Algorithm to calculate input for jet flavour tagging. The algorithm returns the probability that all tracks in the seeded vertex are consistent with being generated at the same vertex. The input DecayChain is expected to be the output of vertex_lcfi::TrackAttach algorithm. In this algorithm all the tracks are either added (attached) to the seed vertex or removed from the DecayChain. The algorithm has the following input parameters:

Parameters

Chisquarecut	- cut on the chi squared of the seed Vertex.
Ntrackscut	- cut on the minimum number of tracks in the seed Vertex.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.26.2 Member Function Documentation

12.26.2.1 double vertex_lcfi::SecVertexProb::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a Jet.

Calculate the momentum of the Seeded Secondary Vertex.

Parameters

DecayChain	Pointer to the DecayChain to be analysed. This DecayChain is an output from the track attach algorithm in which case it is composed only of one vertex and attached tracks.

Returns

double the probability that a secondary vertex is present in the Jet.

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.26.2.2 string vertex_lcfi::SecVertexProb::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.26.2.3 std::vector<string> vertex_lcfi::SecVertexProb::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.26.2.4 std::vector<string> vertex_lcfi::SecVertexProb::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.26.2.5 void vertex_lcfi::SecVertexProb::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.26.2.6 void vertex_lcfi::SecVertexProb::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.26.2.7 void vertex_lcfi::SecVertexProb::setStringParameter (const string & *Parameter*, const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- secondvertexprob.h

12.27 vertex_lcfi::Track Class Reference

Unique Track representation.

`#include <track.h>`

Public Member Functions

- Track ()

  Default Constuctor.
- Track (Event *Event, const HelixRep &HelixRep, const Vector3 &Momentum, const double &Charge, const SymMatrix5x5 &Cov, std::vector< int > hits, void *TrackNum=0)

  Full Constructor.
- Event * event () const

  Event that this track belongs to.
- TrackState * makeState () const

  Create a TrackState of this track.
- const HelixRep & helixRep () const

  Helix represenation of this track.
- double charge () const

  Track charge.
- const Vector3 & momentum () const

  Track perigee momentum.
- const SymMatrix5x5 & covarianceMatrix () const

  Covariance Matrix.
- double significance (Projection Proj) const

  Significance of the track.
- double signedSignificance (Projection Proj, Jet *MyJet) const

  Signed significance of the track.
- const std::vector< int > & hitsInSubDetectors () const

  Number of hits in each sub detector.
- void * trackingNum () const

  Tracking Number.

12.27.1 Detailed Description

Unique Track representation.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.2

Date

20/09/05

12.27.2 Constructor & Destructor Documentation

12.27.2.1 vertex_lcfi::Track::Track (Event * *Event,* const HelixRep & *HelixRep,* const Vector3 & *Momentum,* const double & *Charge,* const SymMatrix5x5 & *Cov,* std::vector< int > *hits,* void * *TrackNum* = 0)

Full Constructor.

Parameters

Event	Pointer to the Event the track belongs to
HelixRep	Helix representation of the track
Charge	Charge of the track
Cov	Covariance Matrix of the track
TrackNum	Tracking number (default=0)

12.27.3 Member Function Documentation

12.27.3.1 double vertex_lcfi::Track::charge ( ) const

Track charge.

Returns

Charge of the Track

12.27.3.2 const SymMatrix5x5& vertex_lcfi::Track::covarianceMatrix ( ) const

Covariance Matrix.

Returns

SymMatrix5x5 of track parameters covariance

12.27.3.3 Event* vertex_lcfi::Track::event ( ) const

Event that this track belongs to.

Returns

A pointer to this tracks event

12.27.3.4 const HelixRep& vertex_lcfi::Track::helixRep ( ) const

Helix represenation of this track.

Returns

The helix representation of this track

12.27.3.5 const std::vector<int>& vertex_lcfi::Track::hitsInSubDetectors ( ) const [inline]

Number of hits in each sub detector.

Returns

vector of ints

12.27.3.6 TrackState* vertex_lcfi::Track::makeState ( ) const

Create a TrackState of this track.

The memory is allocated using the memory manager event lifetime, so don't delete this pointer!

Returns

A pointer to a new trackstate of this track

12.27.3.7 const Vector3& vertex_lcfi::Track::momentum ( ) const

Track perigee momentum.

Returns

Vector3 of the tracks momentum

12.27.3.8 double vertex_lcfi::Track::signedSignificance (Projection *Proj*, Jet * *MyJet*) const

Signed significance of the track.

Parameters

Proj	Projection to take the significance in
Jet	Jet from which to take the momentum to seed the significance

Returns

double of the significance of the track

12.27.3.9 double vertex_lcfi::Track::significance (Projection *Proj*) const

Significance of the track.

Parameters

Proj	Projection to take the significance in

Returns

double of the significance of the track

12.27.3.10 void* vertex_lcfi::Track::trackingNum ( ) const

Tracking Number.

Returns

racking number implemented as pointer to void

The documentation for this class was generated from the following file:

- track.h

12.28 vertex_lcfi::TrackAttach Class Reference

Track attachment algorithm. Calculates the seed vertex and the Tracks attached to it.

`#include <trackattach.h>`

Inheritance diagram for vertex_lcfi::TrackAttach:

Collaboration diagram for vertex_lcfi::TrackAttach:

Public Member Functions

- string name () const

  Name.

- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- DecayChain * calculateFor (DecayChain *MyDecayChain) const

  Run the algorithm on a jet.

12.28.1 Detailed Description

Track attachment algorithm. Calculates the seed vertex and the Tracks attached to it.

The algorithm firstly chooses the seed vertex, defined as the secondary vertex furthest from the IP and then decides which Tracks to attach to this vertex. This algorithm was originally devised to recover tracks from 1-prong decays that are not assigned to any ZVRES vertex, but may contribute to variables one wishes to reconstruct, such as the mass of the decaying heavy flavour particle in a jet. The algorithm returns a vertex_lcfi::DecayChain with Tracks attached to the seed vertex and all other Tracks removed. The algorithm has the following input parameters:

Parameters

AddAllTracks-FromSecondary	determines whether to include or exclude inner vertices.
LoDCutmin	cut on the minimum L/D of the track to be attached to the seed vertex
LoDCutmax	cut on the maximum L/D of the track to be attached to the seed vertex
Closeapproach-Cut	cut on the maximum of the distance of closest approach of a Track wrt the seed axis

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.28.2 Member Function Documentation

12.28.2.1 DecayChain* vertex_lcfi::TrackAttach::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a jet.

Calculate the Seed vertex with its attached Tracks. It puts these into a Decay Chain object which is then readable to other lcfi classes. This decay chain does not hold any other Tracks but the ones in the seed vertex (including the attached tracks).

Parameters

Pointer	to the DecayChain to be analysed

Returns

Pointer to DecayChain of algorithm result

Implements vertex_lcfi::Algo< DecayChain *, DecayChain * >.

12.28.2.2 string vertex_lcfi::TrackAttach::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, DecayChain * >.

12.28.2.3 std::vector<string> vertex_lcfi::TrackAttach::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, DecayChain * >.

12.28.2.4 std::vector<string> vertex_lcfi::TrackAttach::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, DecayChain * >.

12.28.2.5 void vertex_lcfi::TrackAttach::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.28.2.6 void vertex_lcfi::TrackAttach::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.28.2.7 void vertex_lcfi::TrackAttach::setStringParameter (const string & *Parameter*, const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- trackattach.h

12.29 vertex_lcfi::TrackState Class Reference

Spatial point on a Track.

`#include <trackstate.h>`

Public Member Functions

- ~TrackState ()

  Destructor.
- TrackState (Track *Track)

  Construct from a given track.
- TrackState (const HelixRep &init, const double &cha, const SymMatrix5x5 &cov, Track *tra)

  Construct from track parameters.
- void resetToRef ()

  Reset the track to distance swum = 0.
- void swimDistance (const double s)

  Swim the track a fixed distance.
- void swimToStateNearest (const Vector3 &Point)

  Swim to the point of closest approach to Point.
- void swimToStateNearest (TrackState *const TrackToSwimTo)

  Swim to the point of closest approach to another TrackState.
- void swimToStateNearestXY (const Vector3 &Point)

  Swim to the point of closest approach in the XY plane to Point.
- bool sameTrack (TrackState *const Track) const

  Is this the same track as another TrackState?
- bool sameTrack (Track *const Track) const

  Is this the same track as Track?
- double distanceTo (const Vector3 &Point) const

  Distance from the TrackStates current position to Point.
- double distanceTo2 (const Vector3 &Point) const

  Distance squared from the TrackStates current position to Point.
- double xyDistanceTo (const Vector3 &Point) const

  Distance in the XY plane from the TrackStates current position to Point.
- double chi2 (const Vector3 &Point)

  Calculate this tracks minimum chi squared to Point.
- double chi2_nomove (const Vector3 &Point)

  Calculate this tracks chi squared to Point at the TrackStates current position.
- const Vector3 & position () const

  Current position of the trackstate.
- double phi () const

  Current phi of the trackstate.
- double charge () const

Charge.

- bool isNeutral () const

  Is this track neutral.
- bool isCharged () const

  Is this track charged.
- const SymMatrix2x2 & positionCovarMatrix () const

  Current position covariance matrix of the trackstate (d0,z0)
- const SymMatrix2x2 & inversePositionCovarMatrix () const

  Current inverse position covariance matrix of the trackstate (d0,z0)
- const Matrix3x3 vertexErrorContribution (Vector3 point) const

  The error contribution of this trackstate to a vertex at point.
- const SymMatrix3x3 & positionCovarMatrixXYZ () const

  Current position covariance matrix of the trackstate (x,y,z)
- const SymMatrix3x3 & inversePositionCovarMatrixXYZ () const

  Current inverse position covariance matrix of the trackstate (x,y,z)
- Track * parentTrack () const

  The Track that this TrackState belongs to, (if any)
- void debugOut ()

  Print some info to std::cout.

12.29.1 Detailed Description

Spatial point on a Track.

Tracks in detector space have one Track object, but many TrackState objects.

TrackState objects define a point in space on the track, movement along the path of the track is performed by the swimming methods of TrackState. For details of the parameterisation see Track

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.2

Date

20/09/05

12.29.2 Constructor & Destructor Documentation

12.29.2.1 vertex_lcfi::TrackState::TrackState (Track * *Track*)

Construct from a given track.

Constructs a track state with parameters from the given Track, inital position distance swum=0 (usually the perigee to the ref point)

12.29.2.2 vertex_lcfi::TrackState::TrackState (const HelixRep & *init,* const double & *cha,* const SymMatrix5x5 & *cov,* Track * *tra*)

Construct from track parameters.

Constructs a track state given parameters, with inital position distance swum=0 (usually the perigee to the ref point)

The documentation for this class was generated from the following file:

- trackstate.h

12.30 TrueAngularJetFlavourProcessor Class Reference

Determine MC Jet Flavour by angular matching of heavy quarks to jets, also determine hadronic and partonic charge of jet.

`#include <TrueAngularJetFlavourProcessor.h>`

Inherits Processor.

12.30.1 Detailed Description

Determine MC Jet Flavour by angular matching of heavy quarks to jets, also determine hadronic and partonic charge of jet.

The processor looks at all the PDG codes of all MC particles and recognises all particles containing b- and c-quarks. It then looks at the momentum of the heavy MC particles and at the momentum of the jets. The association is done by matching heavy flavour hadrons to the jet that is closest in agle. More than one heavy particle can therefore be associated with the same jet. If this happens the jet flavour is the flavour of the first particle in the parent-daughter chain associated with the jet. The pdg code of particle is subsequently used to determine the hadronic charge of the jet and the partonic charge of the heavy particle.

Input - Prerequisites

Needs the collection of MCParticles. Needs the collection of Reconstructed Particles that represent the jets.

Output

It writes to lcio the calculated flavours of the jets. This is stored in a collection of LCIntVec. By default the collection is called TrueJetFlavour.Writes also the PDG of the used particle and the hadronic and the partonic charge. By definition these collections are called: TrueJetPDGCode, TrueJetHadronCharge and TrueJetPartonCharge.

Parameters

MCParticleCol-Name	Name of the MCParticle collection.
JetRPColName	Name of the ReconstructedParticle collection that represents jets.
TrueJetFlavour-Collection	Name of the output collection where the jet flavours will be stored.
TrueJetPDG-CodeCollection	Name of the output collection where the PDG of the heavy particle associated to the jet is stored.
TrueJetHadron-Charge-Collection	Name of the output collection where the hadronic charge is stroed.
TrueJetParton-Charge-Collection	Name of the output collection where the parton charge (charmness, bottomnes) is stored.
MaximumAngle	Maximum value allowed between MCParticle and jet momentum expressed in degrees. If the closest jet is at a wider angle than MaximumAngle the MC particle does not get assigned.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk),
interface by Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

The documentation for this class was generated from the following file:

- TrueAngularJetFlavourProcessor.h

12.31 vertex_lcfi::TwoTrackPid Class Reference

Simple two track PID for gamma and Ks.

`#include <twotrackpid.h>`

Inheritance diagram for vertex_lcfi::TwoTrackPid:

Collaboration diagram for vertex_lcfi::TwoTrackPid:

Public Member Functions

- string name () const

  Name.
- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- std::map< PidCutType,
  std::vector< Track * > > calculateFor (Jet *MyJet) const

  Run the algorithm on a Jet.

12.31.1 Detailed Description

Simple two track PID for gamma and Ks.

This algorithm calulates the mass of all combinations of Track pairs with opposite charges. If the tracks mass is consistent with the processes gamma ->ee or Ks -> pi+pi- the tracks are flagged and a pointer is inserted into the map under the correct key (either gamma or Ks) The parameters of this algorithm are:

Parameters

MaxGamma-Mass	upper limit on the photon mass
MinKsMass	lower limit on the Ks mass
MaxKsMass	upper limit on the Ks mass
Chi2Cut	cut on the Chi squared for the two tracks beinig in the same vertex.
RPhiCut	cut on the maximum RPhi of the vertex that the tracks form.
SignificanceCut	cut on the minimum rphi significance of the tracks with respect to the IP.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.31.2 Member Function Documentation

12.31.2.1 std::map<PidCutType, std::vector<Track*> > vertex_lcfi::TwoTrackPid::calculateFor (Jet * *MyJet*) const [virtual]

Run the algorithm on a Jet.

Calculates whether two tracks are consistent with the hypothesis of gamma-> ee or with the hypothesis gamma->pi pi . In case the tracks are consisten with either of these hypothesis the algorithm puts a pointer to these tracks into the correct key of the resulting map.

Parameters

Jet	Pointer to Jet to be analysed

Returns

map containing two keys Gamma and Ks which hold the tracks that satisfy these criteria.

Implements vertex_lcfi::Algo< Jet *, std::map< PidCutType, std::vector< vertex_lcfi::Track * > > >.

12.31.2.2 string vertex_lcfi::TwoTrackPid::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< Jet *, std::map< PidCutType, std::vector< vertex_lcfi::Track * > > >.

12.31.2.3 std::vector<string> vertex_lcfi::TwoTrackPid::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< Jet *, std::map< PidCutType, std::vector< vertex_lcfi::Track * > > >.

12.31.2.4 std::vector<string> vertex_lcfi::TwoTrackPid::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< Jet *, std::map< PidCutType, std::vector< vertex_lcfi::Track * > > >.

12.31.2.5 void vertex_lcfi::TwoTrackPid::setDoubleParameter (const string & *Parameter,* const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.31.2.6 void vertex_lcfi::TwoTrackPid::setPointerParameter (const string & *Parameter,* void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.31.2.7 void vertex_lcfi::TwoTrackPid::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- twotrackpid.h

12.32 vertex_lcfi::util::Vector3 Class Reference

Multi-Perpose 3 Vector.

`#include <vector3.h>`

Inherits bounded_vector< double, 3 >.

12.32.1 Detailed Description

Multi-Perpose 3 Vector.

Simple 3 Vector class

Author

Dave Bailey
Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vector3.h

12.33 vertex_lcfi::Vertex Class Reference

Vertex.

`#include <vertex.h>`

Public Member Functions

- Vertex ()

  Default Constuctor.
- Vertex (Event *Event, const std::vector< Track * > &Tracks, const Vector3 &Position, const SymMatrix3x3 &PosError, bool IsPrimary, double Chi2, double Probability, std::map< Track *, double > ChiTrack)

  Full Constructor.
- Vertex (Event *Event, const std::vector< Track * > &Tracks, const Vector3 &Position, const SymMatrix3x3 &PosError, bool IsPrimary, double Chi2, double Probability)

  Almost Full Constructor.
- Vertex (ZVTOP::CandidateVertex *CandidateVertex, Event *Event)

  Construct from Candidate Vertex.
- Event * event () const

  Event.
- const std::vector< Track * > & tracks () const

  Get Tracks.
- bool isPrimary () const

  Is this vertex primary.
- bool & isPrimary ()

  Is this vertex primary.
- void addTrack (Track *AddTrack)

  Add Track.
- bool removeTrack (Track *RTrack)

  Remove Track.
- bool hasTrack (Track *HTrack) const

  Does vertex contain this Track?
- double charge () const

  Charge.
- Vector3 momentum () const

  Momentum.
- const Vector3 & position () const

  Position.
- const SymMatrix3x3 & positionError () const

  Position.
- double chi2 () const

  Chi Squared.
- std::map< Track *, double > chi2OfTracks () const

  Chi Squared Of Tracks.
- double probability () const

  Probability.
- double radius (Projection Proj) const

  Radius.
- double radiusError (Projection Proj) const

  Radius Error.
- double distanceToVertex (Vertex *FarVertex, Projection Proj) const

  Distance to another vertex.
- double distanceToVertexError (Vertex *FarVertex, Projection Proj) const

  Error of distance to another vertex.

12.33.1 Detailed Description

Vertex.

Description

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

12.33.2 Constructor & Destructor Documentation

12.33.2.1 vertex_lcfi::Vertex::Vertex (Event * *Event*, const std::vector< Track * > & *Tracks*, const Vector3 & *Position*, const SymMatrix3x3 & *PosError*, bool *IsPrimary*, double *Chi2*, double *Probability*, std::map< Track *, double > *ChiTrack*)

Full Constructor.

Parameters

Event	Pointer to vertices Event
Tracks	Vector of pointers to tracks that form the vertex
Position	Vector3 of the vertices positon
PosError	SymMatrix3x3 of the vertices error
ChiTrack	map of doubles with Track* key holding the chi squared contribution of every track to the vertex

12.33.2.2 vertex_lcfi::Vertex::Vertex (Event * *Event*, const std::vector< Track * > & *Tracks*, const Vector3 & *Position*, const SymMatrix3x3 & *PosError*, bool *IsPrimary*, double *Chi2*, double *Probability*)

Almost Full Constructor.

Parameters

Event	Pointer to vertices Event
Tracks	Vector of pointers to tracks that form the vertex
Position	Vector3 of the vertices positon
PosError	SymMatrix3x3 of the vertices error

12.33.2.3 vertex_lcfi::Vertex::Vertex (ZVTOP::CandidateVertex * *CandidateVertex*, Event * *Event*)

Construct from Candidate Vertex.

Parameters

CandidateVertex	Vertex to copy from
Event	Event to which the Vertex belongs

12.33.3 Member Function Documentation

12.33.3.1 void vertex_lcfi::Vertex::addTrack (Track * *AddTrack*) [inline]

Add Track.

Add a track to the vertex. Does not affect fit. Will not be added if duplicate

Parameters

AddTrack	Pointer to track to add to vertex

12.33.3.2 double vertex_lcfi::Vertex::charge ( ) const

Charge.

Sum charge of tracks in the vertex

Returns

Track charge sum

12.33.3.3 double vertex_lcfi::Vertex::chi2 ( ) const [inline]

Chi Squared.

Returns

Double of the vertices chi^2

12.33.3.4 std::map<Track*,double> vertex_lcfi::Vertex::chi2OfTracks ( ) const [inline]

Chi Squared Of Tracks.

Returns

map of doubles with Track* key of each tracks chi^2 contribution

12.33.3.5 double vertex_lcfi::Vertex::distanceToVertex (Vertex * *FarVertex*, Projection *Proj*) const

Distance to another vertex.

Distance from the vertex postion to the other vertices position

Parameters

FarVertex	Pointer to the other vertex
Proj	Specifies projection of the distance to be taken

Returns

Distance from the vertex postion to the other vertices position

12.33.3.6 double vertex_lcfi::Vertex::distanceToVertexError (Vertex * *FarVertex*, Projection *Proj*) const

Error of distance to another vertex.

Error of the distance from the vertex postion to the other vertices position, taking into account the uncertainty in both vertices

Parameters

FarVertex	Pointer to the other Vertex
Proj	Specifies projection of radius to be taken

Returns

Error of distance from the vertex to the Eevent's interaction point

12.33.3.7 Event* vertex_lcfi::Vertex::event ( ) const [inline]

Event.

Returns

Pointer to the vertices event

12.33.3.8 bool vertex_lcfi::Vertex::hasTrack (Track * *HTrack*) const

Does vertex contain this Track?

Parameters

HTrack	Pointer to track to check vertex for

Returns

1 if Track found 0 if not

12.33.3.9 bool vertex_lcfi::Vertex::isPrimary ( ) const [inline]

Is this vertex primary.

Returns

bool flag

12.33.3.10 bool& vertex_lcfi::Vertex::isPrimary ( ) [inline]

Is this vertex primary.

Returns

bool flag

12.33.3.11 Vector3 vertex_lcfi::Vertex::momentum ( ) const

Momentum.

Average perigee momentum of the tracks in the vertex

Returns

Vector3 of Average momentum

12.33.3.12 const Vector3& vertex_lcfi::Vertex::position ( ) const [inline]

Position.

Returns

Vector3 of vertex position

12.33.3.13 const SymMatrix3x3& vertex_lcfi::Vertex::positionError ( ) const [inline]

Position.

Returns

SymMatrix3x3 of the error of vertex position

12.33.3.14 double vertex_lcfi::Vertex::probability ( ) const [inline]

Probability.

Returns

Double of the vertices probability

12.33.3.15 double vertex_lcfi::Vertex::radius (Projection *Proj*) const

Radius.

Distance from the vertex postion to the event's interaction point

Parameters

Proj	Specifies projection of radius to be taken

Returns

Distance from the vertex to the event's interaction point

12.33.3.16 double vertex_lcfi::Vertex::radiusError (Projection *Proj*) const

Radius Error.

Error of the distance from the vertex postion to the event's interaction point, taking into account the uncertainty in both the vertex and interaction point

Parameters

Proj	Specifies projection of radius to be taken

Returns

Error of distance from the vertex to the event's interaction point

12.33.3.17 bool vertex_lcfi::Vertex::removeTrack (Track * *RTrack*)

Remove Track.

Remove a track from the vertex. Does not affect fit. Does nothing if track is not in vertex

Parameters

RTrack	Pointer to track to remove from vertex

Returns

1 if Track removed 0 if not found

12.33.3.18 const std::vector<Track*>& vertex_lcfi::Vertex::tracks ( ) const [inline]

Get Tracks.

Returns

Vector of pointers to Tracks in the vertex (can be empty)

The documentation for this class was generated from the following file:

- vertex.h

12.34 vertex_lcfi::VertexCharge Class Reference

Calculation of the charge of the vertex.

`#include <vertexcharge.h>`

Inheritance diagram for vertex_lcfi::VertexCharge:

Collaboration diagram for vertex_lcfi::VertexCharge:

Public Member Functions

- string name () const

  Name.
- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- double calculateFor (DecayChain *MyDecayChain) const

  Run the algorithm on a jet.

12.34.1 Detailed Description

Calculation of the charge of the vertex.

Simple Jet algorithm used to calculate the charge of the vertex. The input is a DecayChain. This DecayChain is expected to be the output of vertex_lcfi::TrackAttach algorithm. In this algorithm all the tracks are either added (attached) to the seed vertex or removed from the DecayChain. The class does not have any input parameters.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.34.2 Member Function Documentation

12.34.2.1 double vertex_lcfi::VertexCharge::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a jet.

Calculate the charge of the Seed Secondary Vertex.

Parameters

DecayChain	Pointer to the DecayChain to be analysed. This DecayChain is an output from the track attach algorithm in which case it is composed only of one vertex and attached tracks.

Returns

double the charge of the seed secondary vertex with all attached tracks.

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.34.2.2 string vertex_lcfi::VertexCharge::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.34.2.3 std::vector<string> vertex_lcfi::VertexCharge::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.34.2.4 std::vector<string> vertex_lcfi::VertexCharge::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.34.2.5 void vertex_lcfi::VertexCharge::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.34.2.6 void vertex_lcfi::VertexCharge::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.34.2.7 void vertex_lcfi::VertexCharge::setStringParameter (const string & *Parameter*, const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- vertexcharge.h

12.35 VertexChargeProcessor Class Reference

Calculates the Vertex Charge.

`#include <VertexChargeProcessor.h>`

Inherits Processor.

Collaboration diagram for VertexChargeProcessor:

12.35.1 Detailed Description

Calculates the Vertex Charge.

The processor calculated the vertex charge of a Decay Chain by using the tracks and the verteces present in the chain. Two logically slightly different algorithms are used depending on the hypothesis of a B or C quark Vertex. In the B hypothesis we include the inner verteces, in the C we do not include them. This choice is controlled by the parameter ChargeAllSecondaryTracks.

Input

- A collection of ReconstructedParticles that represents the jets in the event (obtained from a jet finder, say SatoruJetFinderProcessor, although in order not to break the reconstruction chain we suggest you run this after the flavour tagging. In this way the LCFI chain remains intact).
- A collection of vertices that contains the per event primary vertices; one for each event. (optional) This collection is filled in the vertex_lcfi::PerEventIPFitter processor.
- A collection of decay chains as filled by the the ZVTOPZVRESProcessor or ZVTOPZVKINProcessor.

Output

The processor writes into the selected lcio output file. All the values calculated by the processor are saved in the same LCFloatVec collection. The default name of the output collection is Charge.Although this is changed in the steering files to something more appropriate, depending on B or C calculation. For more details see the interface documentation.

Parameters

VertexCharge-Collection	collection where results will be stored.
ChargeAll-Secondary-Tracks	include or exclude tracks in the inner vertices for the track attachment.
Charge-Closeapproach-Cut	upper cut on track distance of closest approach to the seed axis in the calculation of the vertex charge variable, used by vertex_lcfi::TrackAttach.

ChargeLoD-Cutmax	Cut determining the maximum L/D for the Charge, used by vertex_lcfi::TrackAttach (when calculating C-Charge)
ChargeLoD-Cutmin	Cut determining the minimum L/D for the Charge, used by vertex_lcfi::TrackAttach (when calculating C-Charge)

Author

Erik Devetak(erik.devetak1@physics.ox.ac.uk)

The documentation for this class was generated from the following file:

- VertexChargeProcessor.h

12.36 vertex_lcfi::VertexDecaySignificance Class Reference

Calculation of Decay Significance.

`#include <decaysignificance.h>`

Inheritance diagram for vertex_lcfi::VertexDecaySignificance:

Collaboration diagram for vertex_lcfi::VertexDecaySignificance:

Public Member Functions

- string name () const
  Name.
- std::vector< string > parameterNames () const
  Parameter Names.
- std::vector< string > parameterValues () const
  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)
  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)
  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)
  Set Pointer Parameter.
- std::map
  < DecaySignificanceType,
  double > calculateFor (DecayChain *MyDecayChain) const
  Run the algorithm on a Jet.

12.36.1 Detailed Description

Calculation of Decay Significance.

Jet flavour tagging parameter algorithm that calculates the most significant decay length between vertices present in the inputted DecayChain. The algorithm calculates the distance between all consecutive vertices (i.e. with respect to the previous Vertex) in the Jet and the error of the distance. The significance is defined as the distance divided by the error. The algorithm then outputs the significance and the length of the most significant decay length. The class does not have any input parameters.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.36.2 Member Function Documentation

12.36.2.1 std::map<DecaySignificanceType, double> vertex_lcfi::VertexDecaySignificance::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a Jet.

Calculate the most significant decay length of the decay chain

Parameters

Jet	Pointer to the DecayChain to be analysed

Returns

map containing the following keys: "Significance", "Distance"

Implements vertex_lcfi::Algo< DecayChain *, std::map< DecaySignificanceType, double > >.

12.36.2.2 string vertex_lcfi::VertexDecaySignificance::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, std::map< DecaySignificanceType, double > >.

12.36.2.3 std::vector<string> vertex_lcfi::VertexDecaySignificance::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, std::map< DecaySignificanceType, double > >.

12.36.2.4 std::vector<string> vertex_lcfi::VertexDecaySignificance::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, std::map< DecaySignificanceType, double > >.

12.36.2.5 void vertex_lcfi::VertexDecaySignificance::setDoubleParameter (const string & *Parameter,* const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.36.2.6 void vertex_lcfi::VertexDecaySignificance::setPointerParameter (const string & *Parameter,* void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.36.2.7 void vertex_lcfi::VertexDecaySignificance::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- decaysignificance.h

12.37 vertex_lcfi::ZVTOP::VertexFinderClassic Class Reference

Vertex Finding object - classic ZVTOP.

`#include <vertexfinderclassic.h>`

12.37.1 Detailed Description

Vertex Finding object - classic ZVTOP.

Very unfinished - A set of tracks that vertex finding is performed on, I originally was going to call this a jet, but then realised this may not always be the case. Subject to change as we think about how to interface ZVTOP Note new mwmory management

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfinderclassic.h

12.38 vertex_lcfi::ZVTOP::VertexFinderGhost Class Reference

Vertex Finding object - classic ZVTOP.

`#include <vertexfinderghost.h>`

12.38.1 Detailed Description

Vertex Finding object - classic ZVTOP.

Perform the ghost track algorithm on a set of tracks

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

10/03/06

The documentation for this class was generated from the following file:

- vertexfinderghost.h

12.39 vertex_lcfi::ZVTOP::VertexFitter Class Reference

Vertex Fitter Interface.

`#include <vertexfitter.h>`

Inherited by vertex_lcfi::ZVTOP::VertexFitterKalman, and vertex_lcfi::ZVTOP::VertexFitterLSM.

12.39.1 Detailed Description

Vertex Fitter Interface.

Pure virtual class interface class, cannot be instantiated.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfitter.h

12.40 vertex_lcfi::ZVTOP::VertexFuncMaxFinder Class Reference

Vertex Function Maximum Finder Interface.

`#include <vertexfuncmaxfinder.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexFuncMaxFinder:

12.40.1 Detailed Description

Vertex Function Maximum Finder Interface.

Pure virtual class interface class, cannot be instantiated.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfuncmaxfinder.h

12.41 vertex_lcfi::ZVTOP::VertexFuncMaxFinderClassicStepper Class Reference

Robust VertexFuncMaxFinder.

`#include <vertexfuncmaxfinderclassicstepper.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexFuncMaxFinderClassicStepper:

Collaboration diagram for vertex_lcfi::ZVTOP::VertexFuncMaxFinderClassicStepper:

12.41.1 Detailed Description

Robust VertexFuncMaxFinder.

Similar to original SLD implementaion, minimises along each axis in turn by stepping till minimum reached. Currently hard wired step size.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfuncmaxfinderclassicstepper.h

12.42 vertex_lcfi::ZVTOP::VertexFunction Class Reference

Vertex Function Interface.

`#include <vertexfunction.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexFunction:

12.42.1 Detailed Description

Vertex Function Interface.

Pure virtual class interface class, cannot be instantiated.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfunction.h

12.43 vertex_lcfi::ZVTOP::VertexFunctionClassic Class Reference

VertexFunction as in ZVTOP paper.

`#include <vertexfunctionclassic.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexFunctionClassic:

Collaboration diagram for vertex_lcfi::ZVTOP::VertexFunctionClassic:

Public Member Functions

- VertexFunctionClassic (std::vector< Track * > &Tracks, const double Kip, const double Kalpha, const Vector3 &JetAxis)

  Constructor.
- VertexFunctionClassic (std::vector< Track * > &Tracks, InteractionPoint *IP, const double Kip, const double Kalpha, const Vector3 &JetAxis)

  Constructor.
- ~VertexFunctionClassic ()

  Destructor.
- double valueAt (const Vector3 &Point) const

  Find the value of the vertex function at Point.
- Matrix3x3 firstDervAt (const Vector3 &Point) const

  Find the spacial derivative of the vertex function at Point (not implemented)
- Matrix3x3 secondDervAt (const Vector3 &Point) const

  Find the 2nd spacial derivative of the vertex function at Point (not implemented)

12.43.1 Detailed Description

VertexFunction as in ZVTOP paper.

Function that implements:

$$V(\mathbf{r}) = \sum_{i=0}^{N} f_i(\mathbf{r}) - \frac{\sum_{i=0}^{N} f_i^2(\mathbf{r})}{\sum_{i=0}^{N} f_i(\mathbf{r})}$$

for the Track and InteractionPoint objects $f_i(\mathbf{r})$ given to it at construction for the Vector3 $\mathbf{r}$ specifed.

The above formula is modified thus by additional (Kip, Kalpha, JetAxis) inputs:

$$V(\mathbf{r}) = K_{IP} f_0(\mathbf{r}) + \sum_{i=1}^{N} f_i(\mathbf{r}) - \frac{K_{IP} f_0^2(\mathbf{r}) + \sum_{i=1}^{N} f_i^2(\mathbf{r})}{K_{IP} f_0(\mathbf{r}) + \sum_{i=1}^{N} f_i(\mathbf{r})}$$

K_{IP} is therefore just a weight on the InteractionPoint

$$V(\mathbf{r}) \rightarrow V(\mathbf{r}) exp(-K_\alpha \alpha^2)$$

Where α is the angle between JetAxis and $\mathbf{r}$. K_{IP} is then just a weight on the Jet Axis.

Todo Derivative functions not yet implemented.

This class constucts GaussTube and GaussEllipsoid objects and uses thier valueAt(Vector3 Point) to perform the evaluation, how the tubes are evaluated depends on them.

Destruction cleans up all GaussTube and GaussEllipsoid objects created.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

12.43.2 Constructor & Destructor Documentation

12.43.2.1 vertex_lcfi::ZVTOP::VertexFunctionClassic::VertexFunctionClassic (std::vector< Track * > & *Tracks,* const double *Kip,* const double *Kalpha,* const Vector3 & *JetAxis*)

Constructor.

Creates GaussTubes as needed from Tracks

12.43.2.2 vertex_lcfi::ZVTOP::VertexFunctionClassic::VertexFunctionClassic (std::vector< Track * > & *Tracks,* InteractionPoint * *IP,* const double *Kip,* const double *Kalpha,* const Vector3 & *JetAxis*)

Constructor.

Creates GaussTubes and GaussEllipsoids as needed from Tracks and IP

The documentation for this class was generated from the following file:

- vertexfunctionclassic.h

12.44 vertex_lcfi::ZVTOP::VertexFunctionElement Class Reference

Vertex Fuction Element (Tubes, ellipse) Interface.

`#include <vertexfunctionelement.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexFunctionElement:

12.44.1 Detailed Description

Vertex Fuction Element (Tubes, ellipse) Interface.

Pure virtual class interface class, cannot be instantiated.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfunctionelement.h

12.45 vertex_lcfi::ZVTOP::VertexFunctionSimple Class Reference

Simplified VertexFunction.

`#include <vertexfunctionsimple.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexFunctionSimple:

Collaboration diagram for vertex_lcfi::ZVTOP::VertexFunctionSimple:

12.45.1 Detailed Description

Simplified VertexFunction.

Function that implements

$$V(\mathbf{r}) = \sum_{i=0}^{N} f_i(\mathbf{r}) - \frac{\sum_{i=0}^{N} f_i^2(\mathbf{r})}{\sum_{i=0}^{N} f_i(\mathbf{r})}$$

for the Track and InteractionPoint objects given to it at construction. No Kip,Kalpha modifications.

Todo Derivative functions not yet implemented.

This class constucts GaussTube and GaussEllipsoid objects and uses thier valueAt(Vector3 Point) to perform the evaluation, how the tubes are evaluated depends on them.

Destruction cleans up all GaussTube and GaussEllipsoid objects created.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.1

Date

20/09/05

The documentation for this class was generated from the following file:

- vertexfunctionsimple.h

12.46 vertex_lcfi::VertexMass Class Reference

Calculation of the Pt corrected mass vertex flavour tag input.

`#include <vertexmass.h>`

Inheritance diagram for vertex_lcfi::VertexMass:

Collaboration diagram for vertex_lcfi::VertexMass:

Public Member Functions

- VertexMass ()
  Default Constructor.
- string name () const
  Name.
- std::vector< string > parameterNames () const
  Parameter Names.
- std::vector< string > parameterValues () const
  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)
  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)
  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)
  Set Pointer Parameter.
- double calculateFor (DecayChain *MyDecayChain) const
  Run the algorithm on a jet.

Static Public Member Functions

- static double Ptcalc (Vertex *, Vertex *, Vector3 *, float)
  Calculate vertex-constrained Pt correction.

12.46.1 Detailed Description

Calculation of the Pt corrected mass vertex flavour tag input.

This class calculates the Pt corrected mass flavour tag input. The input is a DecayChain. This DecayChain is expected to be the output of vertex_lcfi::TrackAttach algorithm. In this algorithm all the tracks are either added (attached) to the seed vertex or removed from the DecayChain. The first step of the algorithm is to search the DecayChain for the seed vertex, defined as vertex furthest from the IP. The routine then assumes that there is only one vertex in this DecayChain and that all the tracks in the DecayChain are attached to this vertex. The algorithm

attempts to calculate the mass of this vertex. For this calculation we use the assumption that each track in the vertex has the mass of a charged pion. Additionally a kinematic correction is applied in order to minimise the transverse momentum of the tracks attached to the vertex. This accounts for the fact that some tracks might be neutral particles. There is however a limit on the maximum number of sigmas (based on error matrices) within which the the vertex axis can move. We then use this transverse momentum to calculate the pt corected vertex mass.

Parameters

MaxMomentum-Angle	maximum value of p^2 (1- cos^2(theta)), where theta is the angle between the vertex axis and the total momentum of the tracks attached to the vertex.
MaxKinematic-CorrectionSigma	maximum number of sigmas (based on error matrices) within which the the vertex axis can move.
MaxMomentum-Correction	Maximum factor, by which vertex mass can be corrected (if this value is 2 the maximum mass returned will be twice the mass calculated without kinematic corrections.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.46.2 Member Function Documentation

12.46.2.1 double vertex_lcfi::VertexMass::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a jet.

Calculate the Momentum Corrected mass of the seed vertex.

Parameters

DecayChain	Pointer to the decay chain to be analysed. This decay chain is an output from the track attach algorithm in which case it is composed only of one vertex and attached tracks.

Returns

double: Momentum corrected mass

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.46.2.2 string vertex_lcfi::VertexMass::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.46.2.3 std::vector<string> vertex_lcfi::VertexMass::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.46.2.4 std::vector<string> vertex_lcfi::VertexMass::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.46.2.5 void vertex_lcfi::VertexMass::setDoubleParameter (const string & *Parameter,* const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.46.2.6 void vertex_lcfi::VertexMass::setPointerParameter (const string & *Parameter,* void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.46.2.7 void vertex_lcfi::VertexMass::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- vertexmass.h

12.47 vertex_lcfi::VertexMomentum Class Reference

Calculation of Vertex Momentum.

`#include <vertexmomentum.h>`

Inheritance diagram for vertex_lcfi::VertexMomentum:

Collaboration diagram for vertex_lcfi::VertexMomentum:

Public Member Functions

- string name () const

  Name.

- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- double calculateFor (DecayChain *MyDecayChain) const

  Run the algorithm on a decay chain.

12.47.1 Detailed Description

Calculation of Vertex Momentum.

Jet flavour tagging parameter algorithm that returns the momentum of the seed vertex. The input is a DecayChain. This DecayChain is expected to be the output of vertex_lcfi::TrackAttach algorithm. In this algorithm all the tracks are either added (attached) to the seed vertex or removed from the DecayChain. The class does not have any input parameters.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.47.2 Member Function Documentation

12.47.2.1 double vertex_lcfi::VertexMomentum::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a decay chain.

Calculate the momentum of the Seed Secondary Vertex.

Parameters

DecayChain	Pointer to the decay chain to be analysed. This decay chain is an output from the track attach algorithm in which case it is composed only of one vertex and attached tracks.

Returns

double the momentum of the seed secondary vertex with all attached tracks.

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.47.2.2 string vertex_lcfi::VertexMomentum::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.47.2.3 std::vector<string> vertex_lcfi::VertexMomentum::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.47.2.4 std::vector<string> vertex_lcfi::VertexMomentum::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, double >.

12.47.2.5 void vertex_lcfi::VertexMomentum::setDoubleParameter (const string & *Parameter,* const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.47.2.6 void vertex_lcfi::VertexMomentum::setPointerParameter (const string & *Parameter,* void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.47.2.7 void vertex_lcfi::VertexMomentum::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- vertexmomentum.h

12.48 vertex_lcfi::VertexMultiplicity Class Reference

Calculation of the number of tracks in secondary vertices.

`#include <vertexmultiplicity.h>`

Inheritance diagram for vertex_lcfi::VertexMultiplicity:

Collaboration diagram for vertex_lcfi::VertexMultiplicity:

Public Member Functions

- string name () const
  Name.
- std::vector< string > parameterNames () const
  Parameter Names.
- std::vector< string > parameterValues () const
  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)
  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)
  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)
  Set Pointer Parameter.
- int calculateFor (DecayChain *MyDecayChain) const
  Run the algorithm on a jet.

12.48.1 Detailed Description

Calculation of the number of tracks in secondary vertices.

!Algorithm to calculate input for jet flavour tagging. The algorithm returns the number of tracks in all non primary vertices. The input is a DecayChain. The class does not have any input parameters.

Author

Erik Devetak (e.devetak1@physics.ox.ac.uk)

12.48.2 Member Function Documentation

12.48.2.1 int vertex_lcfi::VertexMultiplicity::calculateFor (DecayChain * *MyDecayChain*) const [virtual]

Run the algorithm on a jet.

Calculate the vertex multeplicity flavour tag

Parameters

DecayChain	Pointer to the decay chain to be analysed.

Returns

int the number of tracks in the non primary vertices.

Implements vertex_lcfi::Algo< DecayChain *, int >.

12.48.2.2 string vertex_lcfi::VertexMultiplicity::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< DecayChain *, int >.

12.48.2.3 std::vector<string> vertex_lcfi::VertexMultiplicity::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< DecayChain *, int >.

12.48.2.4 std::vector<string> vertex_lcfi::VertexMultiplicity::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< DecayChain *, int >.

12.48.2.5 void vertex_lcfi::VertexMultiplicity::setDoubleParameter (const string & *Parameter,* const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.48.2.6 void vertex_lcfi::VertexMultiplicity::setPointerParameter (const string & *Parameter,* void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.48.2.7 void vertex_lcfi::VertexMultiplicity::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- vertexmultiplicity.h

12.49 vertex_lcfi::ZVTOP::VertexResolver Class Reference

Vertex Resolver Interface.

`#include <vertexresolver.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexResolver:

12.49.1 Detailed Description

Vertex Resolver Interface.

Pure virtual class interface class, cannot be instantiated.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.2

Date

01/11/05

The documentation for this class was generated from the following file:

- vertexresolver.h

12.50 vertex_lcfi::ZVTOP::VertexResolverEqualSteps Class Reference

VertexResolver as in ZVTOP paper.

`#include <vertexresolverequalsteps.h>`

Inheritance diagram for vertex_lcfi::ZVTOP::VertexResolverEqualSteps:

Collaboration diagram for vertex_lcfi::ZVTOP::VertexResolverEqualSteps:

12.50.1 Detailed Description

VertexResolver as in ZVTOP paper.

Returns true if the two points can be considered to be resolved. Using the method described in the orginal ZVTOP paper. To find the minimum of the VertexFunction between the two points a crude but robust method is used. This simply looks at a discrete set of points along the line between the points and picks the minimum. The number of sample points is currently hard-wired.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Version

0.2

Date

01/11/05

The documentation for this class was generated from the following file:

- vertexresolverequalsteps.h

12.51 vertex_lcfi::ZVKIN Class Reference

Algorithm interface for decay chain construction or vertexing.

`#include <zvkin.h>`

Inheritance diagram for vertex_lcfi::ZVKIN:

Collaboration diagram for vertex_lcfi::ZVKIN:

Public Member Functions

- ZVKIN ()

  Default Constructor.
- string name () const

  Name.
- std::vector< string > parameterNames () const

  Parameter Names.
- std::vector< string > parameterValues () const

  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)

  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)

  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)

  Set Pointer Parameter.
- DecayChain * calculateFor (Jet *MyJet) const

  Run the algorithm on a jet.

12.51.1 Detailed Description

Algorithm interface for decay chain construction or vertexing.

Description

12.51.2 Member Function Documentation

12.51.2.1 DecayChain* vertex_lcfi::ZVKIN::calculateFor (Jet * *MyJet*) const [virtual]

Run the algorithm on a jet.

Calculate the DecayChain of the jet

Parameters

Jet	Pointer to jet to be analysed

Returns

Pointer to DecayChain of algorithm result

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.51.2.2 string vertex_lcfi::ZVKIN::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.51.2.3 std::vector<string> vertex_lcfi::ZVKIN::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.51.2.4 std::vector<string> vertex_lcfi::ZVKIN::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.51.2.5 void vertex_lcfi::ZVKIN::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.51.2.6 void vertex_lcfi::ZVKIN::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.51.2.7 void vertex_lcfi::ZVKIN::setStringParameter (const string & *Parameter*, const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- zvkin.h

12.52 vertex_lcfi::ZVRES Class Reference

Algorithm interface for decay chain construction or vertexing.

`#include <zvres.h>`

Inheritance diagram for vertex_lcfi::ZVRES:

Collaboration diagram for vertex_lcfi::ZVRES:

Public Member Functions

- ZVRES ()
  Default Constructor.
- string name () const
  Name.
- std::vector< string > parameterNames () const
  Parameter Names.
- std::vector< string > parameterValues () const
  Parameter Values.
- void setStringParameter (const string &Parameter, const string &Value)
  Set String Parameter.
- void setDoubleParameter (const string &Parameter, const double Value)
  Set Double Parameter.
- void setPointerParameter (const string &Parameter, void *Value)
  Set Pointer Parameter.
- DecayChain * calculateFor (Jet *MyJet) const
  Run the algorithm on a jet.

12.52.1 Detailed Description

Algorithm interface for decay chain construction or vertexing.

Description

12.52.2 Member Function Documentation

12.52.2.1 DecayChain* vertex_lcfi::ZVRES::calculateFor (Jet * *MyJet*) const [virtual]

Run the algorithm on a jet.

Calculate the DecayChain of the jet

Parameters

Jet	Pointer to jet to be analysed

Returns

Pointer to DecayChain of algorithm result

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.52.2.2 string vertex_lcfi::ZVRES::name ( ) const [virtual]

Name.

String name of the algorithm

Returns

String name

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.52.2.3 std::vector<string> vertex_lcfi::ZVRES::parameterNames ( ) const [virtual]

Parameter Names.

A vector of the names of the algorithms parameters

Returns

vector of string names

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.52.2.4 std::vector<string> vertex_lcfi::ZVRES::parameterValues ( ) const [virtual]

Parameter Values.

A vector of the values of the algorithms parameters, in the same order as parameter names

Returns

vector of string values

Implements vertex_lcfi::Algo< Jet *, DecayChain * >.

12.52.2.5 void vertex_lcfi::ZVRES::setDoubleParameter (const string & *Parameter*, const double *Value*)

Set Double Parameter.

Set a double parameter

Parameters

Parameter	String of parameter name
Value	double of parameter value

12.52.2.6 void vertex_lcfi::ZVRES::setPointerParameter (const string & *Parameter*, void * *Value*)

Set Pointer Parameter.

Set a pointer parameter

Parameters

Parameter	String of parameter name
Value	pointer to void

12.52.2.7 void vertex_lcfi::ZVRES::setStringParameter (const string & *Parameter,* const string & *Value*)

Set String Parameter.

Set a string parameter

Parameters

Parameter	String of parameter name
Value	String of parameter value

The documentation for this class was generated from the following file:

- zvres.h

12.53 ZVTOPZVKINProcessor Class Reference

Find vertices in a jet using kinematic ZVTOP-ZVKIN algorithm.

`#include <ZVTOPZVKINProcessor.h>`

Inherits Processor.

Collaboration diagram for ZVTOPZVKINProcessor:

12.53.1 Detailed Description

Find vertices in a jet using kinematic ZVTOP-ZVKIN algorithm.

Input

Name	Type	Represents
JetRPCollectionName	ReconstructedParticle	Jets to be Vertexed (eg from SatoruJetFinderProcessor
IPVertexCollectionName	Vertex	Event Interaction Point (eg from PerEventIPFitterProcessor) - optional can be manually specified

Output

Name	Type	Represents
DecayChainCollectionName	ReconstructedParticle	Decay Chains (set of found vertices)
VertexCollection	Vertex	Found vertices
DecayChainRPTracksCollection-Name	ReconstructedParticle	Tracks used in Decay Chains and found vertices

Description

This processor finds vertices in a set of LCIO ReconstructedParticles (usually a jet) using the algorithm ZVTOP(-ZVKIN) Also see (INSERT LINK TO ZVTOP DOC) To be used each ReconstructedParticle must have an attached LCIO Track. Note it is imperative that the tracks have well formed and preferably accurate covariance matrices in d0 and z0. If the covariances are too small fake or no vertices may be found. Too large and vertices will be combined.

The algorithm also requires an interaction point in the form of an LCIO Vertex or a manually set position and covariance. - NOTE Only an ip at the origin (0,0,0) is supported as the ghosttrack has that origin, this will hopefully

be upgraded in a future release.

The set of vertices forming a decay chain as output as set of LCIO Vertices and LCIO ReconstructedParticles for details see the interface documentation

For more details on algorihmic parameters see the ZVKIN paper "zvkin.ps" in the doc directory.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Parameters

JetRPCollection	Name of the ReconstructedParticle collection that represents jets
IPVertex-Collection	Name of the Vertex collection that contains the primary vertex (Optional)
DecayChainRP-TracksCollection-Name	Name of the ReconstructedParticle collection that represents tracks in output decay chains
VertexCollection	Name of the Vertex collection that contains found vertices
DecayChain-CollectionName	Name of the ReconstructedParticle collection that holds RPs representing output decay chains
ManualIPVertex	If false then the primary vertex from VertexCollection is used
ManualIPVertex-Position	Manually set position of the primary vertex (cm) - non origin IP not yet fully supported
ManualIPVertex-Error	Manually set error matrix of the primary vertex (cm) (lower symmetric)
Minimum-Probability	If a vertex candidate has a probability below this it will not be considered - lower value results in more merging and lower vertex multiplicity
InitialGhostWidth	Width in cm of the ghost inital ghosttrack also the smallest width it is allowed to have
MaxChi2Allowed	The ghost track is widened until all forward jet tracks have a chi squared lower than this value
OutputTrackChi2	If true the chi squared contributions of tracks to vertices is written to LCIO

The documentation for this class was generated from the following file:

- ZVTOPZVKINProcessor.h

12.54 ZVTOPZVRESProcessor Class Reference

Find vertices in a jet using topological ZVTOP-ZVRES algorithm.

`#include <ZVTOPZVRESProcessor.h>`

Inherits Processor.

Collaboration diagram for ZVTOPZVRESProcessor:

12.54.1 Detailed Description

Find vertices in a jet using topological ZVTOP-ZVRES algorithm.

Input

Name	Type	Represents
JetRPCollectionName	ReconstructedParticle	Jets to be Vertexed (eg from SatoruJetFinderProcessor
IPVertexCollectionName	Vertex	Event Interaction Point (eg from PerEventIPFitterProcessor) - optional can be manually specified

Output

Name	Type	Represents
DecayChainCollectionName	ReconstructedParticle	Decay Chains (set of found vertices)
VertexCollection	Vertex	Found vertices
DecayChainRPTracksCollection-Name	ReconstructedParticle	Tracks used in Decay Chains and found vertices

Description

This processor finds vertices in a set of LCIO ReconstructedParticles (usually a jet) using the algorithm ZVTOP(ZVRES) described in D. Jackson, NIM A388:247-253, 1997 Also see (INSERT LINK TO ZVTOP DOC) To be used each ReconstructedParticle must have an attached LCIO Track. Note it is imperative that the tracks have well formed and preferably accurate covariance matrices in d0 and z0. If the covariances are too small fake or no vertices may be found. Too large and vertices will be combined.

The algorithm also requires an interaction point in the form of an LCIO Vertex or a manually set position and covariance.

The set of vertices forming a decay chain as output as set of LCIO Vertices and LCIO ReconstructedParticles for details see the interface documentation

For more details on algorihmic parameters see the ZVTOP paper.

Author

Ben Jeffery (b.jeffery1@physics.ox.ac.uk)

Parameters

JetRPCollection-Name	Name of the ReconstructedParticle collection that represents jets
IPVertex-CollectionName	Name of the Vertex collection that contains the primary vertex (Optional)
DecayChainRP-TracksCollection-Name	Name of the ReconstructedParticle collection that represents tracks in output decay chains
VertexCollection	Name of the Vertex collection that contains found vertices
DecayChain-CollectionName	Name of the ReconstructedParticle collection that holds RPs representing output decay chains
ManualIPVertex	If false then the primary vertex from VertexCollection is used
ManualIPVertex-Position	Manually set position of the primary vertex (cm)
ManualIPVertex-Error	Manually set error matrix of the primary vertex (cm) (lower symmetric)
IPWeighting	Weight of the IP in the Vertex Function
JetWeighting-EnergyScaling	Scaling factor for Weight of the jet direction in the Vertex Function. Kalpha = Scaling $*$ Jet-Energy
TwoTrackCut	Chi Squared cut for making initial track pairs - chi squared of either track NOT sum
TrackTrimCut	Chi Squared cut for final trimming of tracks from vertices
ResolverCut	Cut to determine if two vertices are resolved
OutputTrackChi2	If true the chi squared contributions of tracks to vertices is written to LCIO

The documentation for this class was generated from the following file:

- ZVTOPZVRESProcessor.h

Index