Table of Contents

Trigger Code For Developers

Note: This is seriously obsolete

How to define an HLT alley?

- The HLT alleys are defined via options, see for example **HltHadConfAlleySequence.opts** at **$HLTSYSROOT/options**. The alley is a sequencer, that contains different **HltAlgorithms**. Most of the HLT alleys uses the HLT generic code, that is, a collection of few generic algorithms that deal with the reconstruction and the filtering. Usually a reconstruction algorithm is followed by a filtering one.
- An HltAlgorithm takes as input Hlt selections and produce an output Hlt selection. Note that the Hlt selections can be stored or taken from the HltSummary, so, in oder words, the HLT code runs in the HltSummary.
- These are the generic HLT algorithms:
    - **HltTrackFilter:** filter a selection of tracks using a list of filters for tracks.
        - some examples of filter: *"PT,>,2000",* or *"IP_PV2D,||>,0.1",* the first one requires that the transverse momentum of the track is greater than 2 GeV; the second filter requires the minimum absolut impact parameter with respect any vertices in the selection *"PV2D",* that is, the primary vertices made with 2D Velo tracks, be greater than 0.1 mm.
        - To know what filters are available, please look at the code at **$HLTCOMMON/src/HltFunctionFactories.cpp**
        - The options are:
            - *HMyTrackFilter.InputSelection = "Tracks0"*
            - *HMyTrackFilter.FilterDescriptor = {"PT,>,2000","IP_PV2D,||>,0.1"}* (note the "," in the string, and "||" for absolute value)
            - *HMyTrackFilter.OutputSelection = "Tracks1"* (if no option, the default it is the name of the algorithm!)
    - **HltVertexFilter:** filter a selection of vertices, using a list of filters for vertices, similar to **HltTrackFilter**.
        - The options are:
            - *HMyVertexFilter.InputSelection = "Vertex0"*
            - *HMyVertexFilter.FilterDescriptor = {"VertexDimuonMass,>,500."}* (the invariant mass, assumming they are muons, of the two tracks should be greather than 500 MeV)
            - *HMyVertexFilter.OutputSelection = "Vertex1"* (if no option, the default is the name of the algorithm)
    - **HltTrackUpgrade:** Takes as input a selection of tracks and call the reconstruction to "upgrade" them adding information from another tracking detector.
        - Types of reconstruction accepted:
            - *TConf* : a muon track, or calo, adding T stations, normally called "L0 Confirmation"
            - *Velo*: RZVelo tracks to Velo 3D
            - *VeloTT*: Velo 3D to VeloTT tracks
            - *Forward:* Velo to Long tracks
        - If a track was previously "upgraded", the reconstruction is not run twice.
        - The options are:
            - *HMyTrackUpgrade.InputSelection = "TrackVelo"*
            - *HMyTrackUpgrade.RecoName = "Forward"*
            - *HMyTrackUpgrade.OutputSelection = "TrackForward"* (again, if no output selection indicated, it takes the name of the algorithm)
    - **HltTrackMatch:** match two track segments into a track

        - ◊ Types of matches considered:
            - · *VeloCalo*: a Velo3D track and a Calo
            - · *VeloT :* a Velo 3D track and a T segment
        - ◊ The options are:
            - · *HMyTrackMatch.InputSelection = "TrackVelo"*
            - · *HMyTrackMatch.InputSelection2 = "TrackT"*
            - · *HMyTrackMatch.MatchName = "VeloT"*
            - · *HMyTrackMatch.OutputSelection = "TrackVeloT"* (again, by default it takes the name of the algorithm)
    - ♦ **HltVertexMaker:** makes a 2 track vertex.
        - ◊ It combines two selections of tracks, into one selection of vertices
        - ◊ the vertices will have one track at least from the first selection of tracks, so track1+track1, and track1+track2, vertices will be procuded, were track1 belong to the first selection of tracks, and track2 belongs to the second selection of tracks.
        - ◊ The vertices will be created if they pass a list of filters based on two tracks. One example of filter is "DOCA,<,0.2", that is the two tracks should have a Distance Of Closest Approach smaller than 0.2 mm.
        - ◊ The options are:
            - · *HMyVertexMaker.InputSelection = "Tracks0"*
            - · *HMyVertexMaker.InputSelection2 = "Tracks1"*
            - · *HMyVertexMaker.FilterDescriptor = {"DOCA,<,0.2"}*
            - · *HMyVertexMaker.OutputSelection = "Vertices"* (again, by default it takes the name of the algorithm)
    - ♦ **HltVertexUpgrade:** "upgrade" the 2 tracks of the vertex, adding to them information from another tracking detector
        - ◊ It one of the tracks has been already "upgraded", the reconstruction it is not run again.
        - ◊ The possible "upgrades" are the same as the HltTrackUpgrade.
        - ◊ The options are:
            - · *HMyVertexUpgrade.InputSelection = "VertexVelo"*
            - · *HMyVertexUpgrade.RecoName = "Forward"*
            - · *HMyVertexUpgrade.OutputSelection = "VertexForward"* (again, by default, it takes the name of the algorithm)
    - ♦ **HltXXXPrepare:** uses a container of objects of type XXX (L0Muon,L0Calo,Track,Vertex), in the TES, as a selection
        - ◊ These algorithms are the entry points of the HLT
        - ◊ All the L0 candidates are converted into object of type Track, for convenience.
        - ◊ In the case of XXX as Track or Vertex, this algorithm is identical as **HltTrackFilter**, except that the input comes from the TES, that means that one can apply a set of filters to the input tracks to make a selection.
        - ◊ The options are (as the HltTrackFilter) except:
            - · *HMyXXXPrepare.TESInput = "Hlt/Track/RZVelo"*
            - · *HMyXXXPrepare.FilterDescriptor = {"IP_PV2D,>,0.1"};* (only for **HltTrackPrepare** and **HltVertexPrepare**)
    - ♦ **HltSelectionFilter:** set a positive decision if any of the input selections has been passed
        - ◊ The options are:
            - · *HMySelectionFilter.InputSelections = {"HltSelection1","HltSelection2"}*
            - · *HMySelectionFiter.OutputSelection = "Selection1_or_Selection2"* (by default takes the name of the algorithm)
    - ♦ **L0Entry**: set a positive decision if any of the L0 channels indicated has a positive decision.
        - ◊ The options are:
            - · *HMyL0Entry.L0ChannelNames = {"Hadron"}* (names as in L0DUReport)
            - · *HMyL0Entry.OutputSelection = "MyL0Entry"* (by default it takes the name of the algorithm)

How to save a Selection in the or the TES?

- You need to indicate the name of the Hlt selection to the algorithm **HltSummaryWriter**,
    - *HltSummaryWriter.Save += {"MySelection"};*
- To save all the Hlt selections in the HltSummary, activate the option:
    - *HltSummaryWriter.SaveAll = true;*
- To copy the list of candidates of a HltSelection into the TES, added the name of the selection to **HltSelectionToTES** algorithm, if the candidates are track, they will show in the TES in the location "Hlt/MySelection/Track"
    - *HltSelectionToTES.Copy = {"MySelection"};*
- To copy the candidates of all Hlt selection into the TES, activate the option:
    - *HltSelectionToTES.CopyAll = true;*

How to monitor the HLT?

- **HltAlgorithm** monitors automatically the number of candidates in the input and output selections. To book the histogram from the options do the following, note the histogram will be stored in the output file under a directory with the name of *MyAlgorithm*
    - *MyAlgorithm.HistogramDescriptor += {"MyInputSelection",("MyInputSelection",20,0.,20.),"MyOutputSelection",("MyOutputSelection",10,0.*
- The **HltTrackFilter** and **HltTrackVertex** algorithms also monitor the quantities in which we apply a filter, i.e " *PT,>,1500",* will produce to histogram, *"PT* " with the distribution of the Pt of all the input candidates, and *"PTBest"* with the distribution, in this case, of the input candidate with the highest Pt value. To book and fill this histogram of the quantities associated to the filters, add the following option:
    - *MyAlgorithm.HistogramDescriptor += {"PT",("PT",100,0.,6000.)};*

How to write a new C++ filter for the HLT?

- The Hlt filter algorithms, **HltTrackFilter**, and **HltVertexFilter**, uses filters,ie *"PT,>,1500.".* The filters rely on a function that compute a quantity, in this case, a PT function.
- There are two types of functions: univaluated, with one argument, as *PT(track),* and bivaluated, with two arguments, as *IP(Track,Vertex)*
- To write a new C++ filter you need to code a function (uni or bivaluated), and to declare it to the factory:
    - To code a C++ function:
        - A simple function must inherit form **Hlt::TrackFunction**, or **Hlt::TrackVertexBiFunction**, and implement the *operator().* To do so, please look at the file **HltFunctions.h, HltFunctions.cpp** at $ **HLTCOMMONROOT/src,** and if possible, place your code in the HltFunction files.
        - To create a function that delegates the computation of the quantity to a **tool**, for example, *"IsMuon",* your tool should inherit from *ITrackFunctionTool.*
    - To decleare your function to the factory (either a simple function, or a function that delegates in a tool), follow the code at **$HLTCOMMONROOT/src/HltFunctionFactories.cpp**

How to write a new C++ HLT Algorithm or Tool?

- In principle, the HLT generic algorithms should cover your needs, if not, you can use the following Hlt base clases, and get inspiration of the Hlt common algorithms code at **$HLTCOMMONROOT/src**:
- **HltBaseAlg:**
    - Provides the basic functionality of the HLT for a simple *algorithm:*

        - ◊ booking histogram via options using the *HistoDecriptor*
            - · A frequency to fill histograms, controlled by the option *HistogramUpdatePeriod*
        - ◊ counters and printouts of the counters
        - ◊ access to all the Hlt selections in the event, their decision and candidates
        - ◊ access to the Hlt configuration, that is the flow of the Hlt, the selections names, and filters strings.
- **HltTool:**
    - Provides the basic functionality of the HLT for a ***tool*** (see above)
- **HltAlgorithm:**
    - An algorithm to produce a Hlt selection as output and that can use Hlt selections as input
    - Provides:
        - ◊ The basic HLT functionality (see HltBaseAlg)
        - ◊ The options for the input selections: *InputSelection, InputSelection2, InputSelections*
        - ◊ The options for the output selection: *OutputSelection* (by default it takes the name of the algorithm)
        - ◊ If the input selections have not a positive decision, or have no candidates, the algorithm execution stops.
        - ◊ The input and output selection candidates are monitored by histograms, that can be booked via the option *HistogramDescriptor* (see above)
        - ◊ If there is more than one output candidate, the output selection is possitive and the Hlt process continues, otherwise stops.
            - · The minimun number of output candidates for a positive decision if controlled by *MinCandidates* option.
    - C++ guide to implement a HltAlgorithm:
        - ◊ You need to retrieve (the input selections) and register (the output selection) at the initialize() method of your algorithm
            - · *m_inputTracks = &retrieveSelection<Track>(m_inputSelectionName);*
            - · *m_outputTracks = &registerSelection<Track>(m_outputSelectionName);*
        - ◊ You need to fill the output selection with the candidates in the execute() method of your algorithm
            - · *m_outputTracks->push_back(mycandidate);*
        - ◊ If you do not produce a list of candidates, only a boolean decision, please add the *decision* true or false in the execute() method
            - · *setDecision(decision);*
        - ◊ In the example above: *m_inputTracks, m_outputTracks* are pointers to **Hlt::TrackSelection** type, defines as members variables in **HltAlgorimth**, and mycandidate is a pointer to **LHCb::Track** type.

How to incorporate tracking reconstruction to the HLT?

- How to upgrade a Track?
    - That is: how to extend a track, adding information from a subdetector, i.e, use a muon track as seed to produce tracks that have also T segments.
    - Implement a **Gaudi tool** with the interface **ITracksFromTrack**
    - Declare your reconstruction tool in the *recoConfiguration()* method of the file **$HLTTRACKINGROOT/src/TracksUpgradeTool.cpp,** following the other cases as example.
- How to match two tracks segments into one track?
    - That is, a velo segment and T segment into a forward track.
    - Implement a **Gaudi tool** with the interface **ITrackMatch**
    - Declare your matching tool in the *recoConfiguration()* method of the file **$HLTTRACKINGROOT/src/TrackMatch.cpp**, following the other cases as example.

How to monitor an alley in Python?

- Get Hlt/HltPython package
- In python/examples there is a python script **hltalleymonitor.py** to monitor an alley
  - ◊ *python hltalleymontitor.py -c Bs2PhiPhi -n 1000 -a SingleHadron*
- Inputs are the channel, the number of events, and the list of alleys names to monitor
- Given the alley name the following monitoring taks can be performed:
  - Rate (or step) plot for the sequencial steps of an alley, rate only on TOS events
  - Candidates for each step of the alley (again possibility of TOS candidates only)
  - Distribution of the variables in which we cut (for all candidates, the best and for TOS only)
- A ROOT file is produce with the histograms related with the monitoring tacks,
  - i.e: SingleHadron:Rate, SingleHadron:RateTOS, SingleHadron:Candidates, etc

For more info please send an email to Jose A. Hernando hernando@cernNOSPAMPLEASE.ch

-- MiriamGandelman - 16 Oct 2006

How to associate HLT fitted tracks

- In order to associate tracks which you have fitted in the HLT, you need to add the corresponding Track container to $HLTCONFROOT/options/HltTrackAssociator.opts.

This topic: LHCb > HltForDevelopers
Topic revision: r12 - 2009-11-25 - PatrickSKoppenburg