Table of Contents

TisTosParticleTagger DVAlgorithm

This algorithm was first added to Phys/TisTosTobbing in v5r0 (use DaVinci v25r7 and higher). Chris Thomas and Vava contributed early prototypes for this code. You can make the code work with older releases - see at the bottom.

There was a major change in the algorithm functionality in Phys/TisTosTobbing v6r7 (DaVinci v28r1p4 or higher) to allow particle filtering via Hlt1, Hlt2 and L0 requirements at once rather than in separate algorithm instances. The change was essentially backwards compatible, except that in the old version mixing Hlt1 and Hlt2 requirements would result in an OR between these specs, while in the new version AND is taken. This version will probably work with any DaVinci release v27r0 or higher (getpack Phys/TisTosTobbing r121071). Here we describe the new functionality, but mark the new features with (new version) .

It has three possible uses:

1. as Particle Filter to pass only Particles which satisfy the TisTos criteria. This feature was not in the original v5r0 release; use v5r1 or higher.
2. as Event filter based on TisTos requirements
3. as Tagger of Particles stored on TES with TisTos results added to ExtraInfo of the Particle. Once such information is added, the tagged Particles can be filtered later on with INFO() Loki Hybrid function. This method is not allowed in Hlt2 or stripping. User can also use such tagging to see which requirements were passed by given Particle if using more than one requirement.

There are two properties which must be set by the user every time the Algorithm is configured; **InputLocations** for location of signal particles (this is done automatically for you if you use Juan's Selection framework) and **TisTosSpecs** for definition of TisTos criteria.

Table of Contents

TisTosSpecs property

It is of type std::map<std::string,int>. The string specifies the "Trigger Input" either via specific trigger line name or via regex expression to be matched against legal trigger names for given event.

The string also specifies TisTos requirement to be imposed, which can be either TOS, TIS or TUS. TUS = "Trigger Used Signal", this condition is satisfied by a trigger candidate which is fully (i.e. TOS) or partially (i.e. TPS) contained within the signal. The Trigger Input and TisTos requriment are separated by "%" character.

The int specifies int-key to be used when saving the TisTos result in ExtraInfo of analyzed Particle. Do not use this feature unless you are using method 3 (can't use it in Hlt & Stripping). Set this number to 0.

If you set any of the int-keys for ExtraInfo storage to a non-zero value, the algorithm will always evaluate all requested TisTos requriments. If all int-keys are zero, the algorithm will skip evaluating subsequent requirements, if the outcome of the Particle filtering is already determined (new version).

```
from Configurables import TisTosParticleTagger
myTagger = TisTosParticleTagger("MyTagger")
myTagger.InputLocations = [ "/Event/Phys/StdLooseJpsi2MuMu/Particles" ]
myTagger.TisTosSpecs = {  L0.*Mu.*Decision%TOS :0, "Hlt1.*Mu.*Decision%TOS":0,
                                    "Hlt1.*Hadron.*Decision%TIS":0,  Hlt2.*Mu.*Decision%TOS :0
DaVinci().UserAlgorithms = [ myTagger ]
```

In this example, Particles which satisfy:

```
( "Hlt1.*Mu.*Decision%TOS" || "Hlt1.*Hadron.*Decision%TIS" ) &&    Hlt2.*Mu.*Decision%TOS  &&
 L0.*Mu.*Decision%TOS
```

will be passed to the output container. (This is true in the new version. In the old version mixing of L0 and Hlt requirements in the same algorithm instance is not allowed. The old version also always takes OR between all requirements.) The reordering of the requirements between the TisTosSpecs and the corresponding logic listed above is intentional. The algorithm will always do L0 requirements last, since they are more expensive in CPU as they require special processing of raw L0 info.

By default setFilterPassed(true) is set only if at least one Particle satisfies the TisTos requirement(s). You can set

```
myTagger.PassOnAll = TRUE
```

to always pass the event independently of TisTos results (this turns the algorithm into pure Particle Tagger with no Event filtering). This also changes which Particles are saved in the output container. By default only the particles which satisfy the TisTos criteria are saved. Once PassOnAll is set to true then all particles are saved (thus the input and output containers become identical). This may be useful if you use method 3.

If the trigger names are given with no regex special characters, you can speed up the algorithm by setting:

```
myTagger.NoRegex = TRUE
```

(an example of regex special characters - ".*").

Using TisTosParticleTagger as particle filter (Method 1)

In this example (run on Dimuon stream) we select /Event/Dimuon/Phys/DiMuonIncLine/Particles which are TOS with respect to Hlt1 DiMuon triggers:

```
from PhysSelPython.Wrappers import AutomaticData, Selection, SelectionSequence

from Configurables import TisTosParticleTagger
_myTagger = TisTosParticleTagger("MyTagger")
_myTagger.TisTosSpecs = { "Hlt1.*DiMuon.*Decision%TOS":0 }
selMyDimuon = Selection("SelDimuonTriggered", Algorithm = _myTagger,
 RequiredSelections =[AutomaticData(Location="/Event/Dimuon/Phys/DiMuonIncLine/Particles")])
selSequence = SelectionSequence( "seqSelDimuonTriggered",  TopSelection=selMyDimuon )

from Configurables import DaVinci
DaVinci().DataType="2010"
DaVinci().UserAlgorithms = [selSequence]
DaVinci().InputType='DST'
```

Using this on MDSTs requires additional set-up options - see here.

See also "Using in Hlt2 context" section.

Filtering on stored TisTos results in Particle ExtraInfo (Method 3)

This is based on tips from Vanya.

The only reason why you would sometimes need to filter via method 3) rather than 1) is that you may want to CombineParticles with criterion less stringent than all Particles satisfying the TisTos criteria (e.g. you may require that at least one satisfies them - see example below).

Use int-keys above 10000 (reserved for users) but not too large to stay within legal int values. If you use "0" then the Particles will not be tagged. It is fine to use the same int-key with more than one "Trigger Input" - in that case the result stored is an OR between all specs for the same int-key. Here is an example of adding TisTosParticleTagger to DaVinci sequence:

```
from Configurables import TisTosParticleTagger
myTagger = TisTosParticleTagger("MyTagger")
myTagger.InputLocations = [ "/Event/Phys/StdLooseJpsi2MuMu" ]
myTagger.TisTosSpecs = { "Hlt1.*Hadron.*Decision%TUS":12345, "Hlt1.*Mu.*Decision%TOS":22246 }
DaVinci().UserAlgorithms = [ myTagger ]
```

Once you execute the Tagger you can filter the tagged Particles with LoKi functor INFO. Here is an example to work together with the example given above. It will select only those StdLooseJpsi2MuMu candidates which were used in Hlt1 Hadron triggers (TUS). The subselection will appear under 'Phys/HadronTUSJPsi'.

```
from Configurables import FilterDesktop
_hadronTUSFilter = FilterDesktop('HadronTUSFilter')
_hadronTUSFilter.Code = " 0 < INFO( 12345, 0.0 ) "

from PhysSelPython.Wrappers import AutomaticData, DataOnDemand, Selection, SelectionSequence
SelStdLooseJPsi = DataOnDemand(Location = "Phys/StdLooseJpsi2MuMu")

hadronTUSJPsi = Selection(name = 'HadronTUSJPsi',
    Algorithm = _hadronTUSFilter,
    RequiredSelections = [ SelStdLooseJPsi])
SomeSelSeq = SelectionSequence('SomeSel',TopSelection = hadronTUSJPsi )
DaVinci().UserAlgorithms += [ SomeSelSeq.sequence() ]
```

You can also filter combinations in CombineParticles:

```
_JPsiAndX = CombineParticles("JPsiAndX")
_JPsiAndX.Preambulo = [ " HadTUS = 0.1 < INFO( 12345, 0.0 ) " ]
_JPsiAndX.Code = " 0 < ANUM( HadTUS ) "
...
```

Here we requested that at least one of the daughter particles is "Hlt1.*Hadron.*Decision%TUS". To require at least two we would have cut:

```
_JPsiAndX.Code = " 1 < ANUM( HadTUS ) "
```

Notice that you can combine Particle selections which were tagged and not tagged with TisTos info here.

In these examples we filtered the Particles which were specified as input to TisTosParticleTagger, thus a mixture of Particles which passed or failed the specified TisTos criteria. You can also filter the output Particle container from the TisTosParticleTagger which (by default) contains only Particles which satisified at least one specified TisTos criterion. When using the output container In case of single TisTos requirement, there is no need to filter on ExtraInfo in a separate step (thus set ExtraInfo index to zero).

Using in Hlt2 context

The TisTosParticleTagger will automatically switch to "TriggerTisTosInHlt" tool instead of offline "TriggerTisTos" when placed in Hlt sequence (it checks OnOffline tool online() method). You can TisTos Hlt1 trigger decisions with respect to Hlt2 particles when using it in Hlt.

Please be aware that if you dial L0 TisTos criteria you will also need to equip your Hlt2 line with decoding of L0 raw banks, which can be quite slow, thus is not advised.

In Hlt2 you are likely to filter on the output container of TisTosParticleTagger (Method 1). For Hlt1 in 1Track configuration you should simply require Hlt1 TOS in your filter (see an example below). If 2-Track triggers make a come back in Hlt1, you can filter twice for the best speed up. First filter single-track Particles to produce Hlt1 TUS (see also "Tightening TUS criterion" below).Then create their combinations and require Hlt1 TOS for the final combination.

Here is an example from Chris Thomas (and Alex Shires) how to use TisTosParticleTagger in Hlt2 code ('unfiltered' is a generic Hlt2 container - e.g. the two-body topo combinatorics).

```
from Configurables import TisTosParticleTagger
TOSInputParticlesFilter = TisTosParticleTagger("TOSInputParticlesFilter")
TOSInputParticlesFilter.TisTosSpecs = { "Hlt1Track.*Decision%TOS":0 }

# keep next 4 lines only if you are sure the Hlt1 trigger candidates do not contain
# (do not use e.g. with Hlt1 Ecal lines)
TOSInputParticlesFilter.ProjectTracksToCalo = FALSE
TOSInputParticlesFilter.CaloClustForCharged = FALSE
TOSInputParticlesFilter.CaloClustForNeutral = FALSE
TOSInputParticlesFilter.TOSFrac = { 4:0.0, 5:0.0 }

# uncomment next line if you use "TUS" filtering instead of "TOS"
# TOSInputParticlesFilter.CompositeTPSviaPartialTOSonly = TRUE

TOSInputParticlesFilter.InputLocations = [unfiltered.outputSelection() ]
_tosfiltered = bindMembers("Filtered", [ unfiltered, TOSInputParticlesFilter ])

# dummy filter needed for Hlt2Line if TisTosParticleTagger is the last filter in the
_filterdesktop = Hlt2Member(FilterDesktop, 'Filter',
       InputLocations=[_tosfiltered], Code="ALL")
_seqTOSfiltered = bindMembers('FilteredWithTOS',[_tosfiltered, _filterdesktop])
```

Using to TisTos L0 trigger

In the new version you simply include L0 TisTos requirements among the trigger specs.

In the old version you must do it in separate algorithm instance and you need to force L0TriggerTisTos tool to be used via:

```
myTagger.TriggerTisTosName= "L0TriggerTisTos"
myTagger.TisTosSpecs = { "L0.*Muon.*Decision%TIS":0 }
```

Using to TisTos any TES container of Tracks, Vertecies or Particles

You need to force TESTisTos tool to be used via:

```
myTagger.TriggerTisTosName= "TESTisTos"
myTagger.TisTosSpecs = { "/Event/Dimuon/Phys/DimuonIncLine/Particles%TOS":0 }
```

Please notice that using this method you can align your Particles with inclusive stripping lines. Just dial their TES location in the "Trigger Input".

Tightening TUS criterion

The default TUS behavior is that TUS is true if any part of the trigger candidate is not TIS (by default <1% of hits shared with signal). This will pass your Hlt2 tracks which have good overlap with e.g. any of the two-Tracks selected by DiHadron line but also all the clones which have little overlap but more than 1%. There is a way to change this behavior and require that TUS is True only if there was good (i.e. TOS-type) overlap with part of the trigger object. For example, let us assume that signal is single Hlt2 track and trigger candidate is a Track-pair selected by Hlt1 DiHadron line. In more restrictive method TUS would be true only if one of the two trigger tracks was TOS (>70% hit sharing) with respect to the Hlt2 track. (Of course, the whole pair will never be TOS with respect to the signal consisting of only one Track). To activate this option do:

```
myTagger.CompositeTPSviaPartialTOSonly = TRUE
```

This can provide even bigger rejections than default TUS.

Making the TisTos tool faster

Considerable amount of CPU is spent in the TisTos tool by collecting calorimeter hits from the signal since this activates projection of Tracks into caloriemeters and Calo reco on demand. If you want to TisTos based on tracking detectors only do:

```
myTagger.ProjectTracksToCalo = FALSE
myTagger.CaloClustForCharged = FALSE
myTagger.CaloClustForNeutral = FALSE
myTagger.TOSFrac = { 4:0.0, 5:0.0 }
```

Executing in DEBUG mode to see the TisTos results in log file

Just putting the algorithm in the DEBUG OutputLevel will start printing a lot of staff from the TriggerTisTos tool itself. To suppress the latter do:

```
myTagger.OutputLevel = 2
from Configurables import TriggerTisTos
myTagger.addTool(TriggerTisTos('TriggerTisTosTool'))
myTagger.TriggerTisTosTool.OutputLevel = 3
# in the new version and using L0 criteria also add:
from Configurables import L0TriggerTisTos
myTagger.addTool(L0TriggerTisTos('L0TriggerTisTosTool'))
myTagger.L0TriggerTisTosTool.OutputLevel = 3
```

.

-- TomaszSkwarnicki - 9-Aug-2010 (Last update 30-Mar-2011)

This topic: LHCb > TisTosParticleTagger
Topic revision: r11 - 2011-03-31 - TomaszSkwarnicki