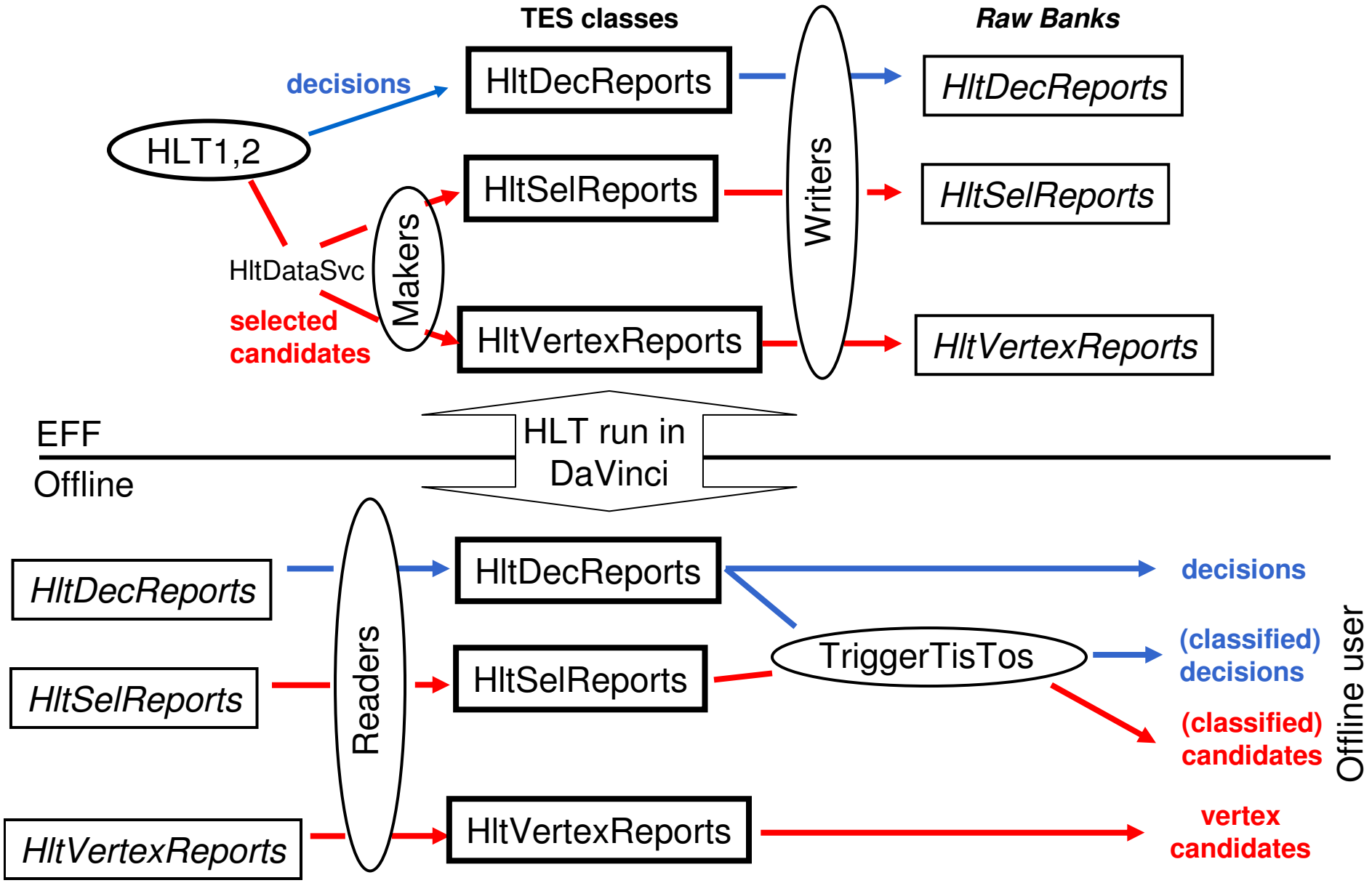
Hlt Raw Data & TISTOS tool (C++ & Python Tutorial)

Tomasz Skwarnicki

Syracuse University

(based on DaVinci v23r1)

Hlt Raw Data



"classified" with respect to offline selected signal

Running HLT in DaVinci

- Setup DaVinci environment

```
unix> SetupProject DaVinci v23r1
```

- To run HLT from DaVinci (here for DC06 MC data) include in your python options:

```
from Configurables import DaVinci
DaVinci().DataType = 'DC06'
DaVinci().Simulation   = True
DaVinci().ReplaceLOBanksWithEmulated = True
DaVinci().HltType = 'Hlt1+Hlt2'
```

Running DaVinci in GaudiPython

- If you are not interested in python ignore this and concentrate on C++ examples
- This is not a python tutorial nor description of all different ways you can use python to analyze LHCb data
  - see previous Software Week tutorials on GaudiPython, Panoramix, Bender
- Minimal python script for running HLT+DaVinci in GaudiPython:

```
from Gaudi.Configuration import *

from Configurables import DaVinci
DaVinci().DataType = "DC06"
DaVinci().Simulation   = True
DaVinci().ReplaceLOBanksWithEmulated = True
DaVinci().HltType = 'Hlt1+Hlt2'

import GaudiPython
appMgr= GaudiPython.AppMgr(outputlevel=3)
sel= appMgr.evtsel()      # to define input file
sel.open(['PFN:/afs/cern.ch/lhcb/group/trigger/vol1/dijkstra/Selections/Bs2MuMu-lum2.dst'])
evt= appMgr.evtsvc()
appMgr.run(1)             # process one event

evt.dump()                # dump different locations on TES (if you wish)
```

Accessing decisions in HltDecReports

C++

```
#include "Event/HltDecReports.h“

// get pointer to HltDecReports from TES
const HltDecReports* decReports =
                get<HltDecReports>(LHCb::HltDecReportsLocation::Default);

// if you are sure trigger has to be there, this is OK
bool hlt1dec = decReports->decReport("Hlt1Global")->decision();

// otherwise check the HltDecReport pointer
const HltDecReport* dihadRep =
                        decReports->decReport("Hlt1DiHadronDecision");
if( dihadRep ){
        info()<< " Hlt1DiHadronDecision " << dihadRep->decision()
        << endmsg;
} else {
        info()<< " Hlt1DiHadronDecision was not configured" << endmsg;
}

// get all trigger names in this configuration
std::vector<std::string> allConfiguredTrgLines
                                        = decReports->decisionNames();

// print entire report container:
  info() << *decReports << endmsg;
```

Accessing decisions in HltDecReports

python

```
# get HltDecReports from TES
decReports = evt['/Event/Hlt/DecReports']

# if you are sure trigger has to be there, this is OK
hlt1dec = decReports.decReport('Hlt1Global').decision()

# otherwise check the HltDecReport pointer
dihadRep = decReports.decReport('Hlt1DiHadronDecision')
if dihadRep != None:
    print ' Hlt1DiHadronDecision ',dihadRep.decision()
else:
    print ' Hlt1DiHadronDecision was not configured '

# get all trigger names in this configuration
allConfiguredTrgLines = decReports.decisionNames()

# print entire report container
print decReports
```

HltDecReports sample dump

- Obtained on $B_s \to \mu\mu$ event in DaVinci v23r1 (many lines not shown here to fit a slide)

```
…    INFO  HltDecReports :  configuredTCK=0 {
decisionName : Hlt1DiHadronDecision HltDecReport : {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 5 intDecisionID : 201 }
decisionName : Hlt1DiMuonIPCL0SegDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 5 intDecisionID : 122 }
decisionName : Hlt1DiMuonNoIP2L0Decision HltDecReport :  {  decision : 1 errorBits : 0 numberOfCandidates : 6 executionStage : 7 intDecisionID : 111 }
decisionName : Hlt1DiMuonNoIPL0DiDecision HltDecReport :  {  decision : 1 errorBits : 2 numberOfCandidates : 2 executionStage : 7 intDecisionID : 110 }
decisionName : Hlt1DiMuonNoIPL0SegDecision HltDecReport :  {  decision : 1 errorBits : 0 numberOfCandidates : 2 executionStage : 7 intDecisionID : 112 }
decisionName : Hlt1Global HltDecReport :  {  decision : 1 errorBits : 0 numberOfCandidates : 0 executionStage : 7 intDecisionID : 1 }
decisionName : Hlt1IgnoringLumiDecision HltDecReport : {  decision : 1 errorBits : 0 numberOfCandidates : 0 executionStage : 7 intDecisionID : 41 }
decisionName : Hlt1L0DiMuon,lowMultDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 2 intDecisionID : 14 }
decisionName : Hlt1L0DiMuonDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 2 executionStage : 6 intDecisionID : 12 }
decisionName : Hlt1L0ElectronDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 2 intDecisionID : 15 }
decisionName : Hlt1LumiDecision HltDecReport : {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 3 intDecisionID : 40 }
decisionName : Hlt1MuTrack4JPsiDecision HltDecReport : {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 0 intDecisionID : 151 }
decisionName : Hlt1MuTrackDecision HltDecReport :  {  decision : 0 errorBits : 2 numberOfCandidates : 0 executionStage : 5 intDecisionID : 150 }
decisionName : Hlt1PhotonDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 2 intDecisionID : 300 }
decisionName : Hlt1VeloClosingDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 3 intDecisionID : 60 }
decisionName : Hlt2Global HltDecReport :  {  decision : 1 errorBits : 0 numberOfCandidates : 0 executionStage : 7 intDecisionID : 2 }
decisionName : Hlt2TopoTF4BodySADEVDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 0 intDecisionID : 50810 }
decisionName : Hlt2UnbiasedBs2PhiPhiDecision HltDecReport :  {  decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 5 intDecisionID : 50135 }
decisionName : Hlt2UnbiasedDiMuonDecision HltDecReport :  {  decision : 1 errorBits : 0 numberOfCandidates : 2 executionStage : 7 intDecisionID : 50200 }
```

Accessible as member functions of HltDecReport

Physics and non-physics trigger lines and Hltx Global reports.
Among physics lines post-scaled L0xxx pass-through-Hlt1 lines.

The value of numberOfCandidates() saturates at 15 (means 15 or more candidates)

(from Gerhard: errorBits=2 in this release means executions of this line took a long time)

HltDecReport – executionStage()

- Meaning of executionStage() values in HltDecReport :

0 initial (did not pass prescale)
1 passed prescale (did not pass L0 seeding)
2 passed seeding (did not pass filter-0)
3 passed filter-0 (did not pass filter-1)
4 passed filter-1 (did not pass filter-2)
5 passed filter-2 (did not pass filter-last)
6 passed filter-last (did not pass postscale)
7 passed postscale [decision is set to true]

- More bits were added to this field – the meaning of the values is likely to change in the future release (max value 255)

Accessing decisions via TriggerTisTos tool

- **In addition to getting decision() of individual trigger lines you can also check logical-OR of decisions between many lines**
  - Particularly easy if you can address all trigger lines of interest with a wild character matching:

C++

```
#include "Kernel/ITriggerTisTos.h"
ITriggerTisTos* m_tisTos;   // in .h file
           // get TisTos tool in initialize():
m_tisTos= tool<ITriggerTisTos>("TriggerTisTos",this);
           // use it in execute():
m_tisTos->setOfflineInput(); // dummy signal
bool hlt1MuDec= m_tisTos->triggerTisTos("Hlt1*Mu*Decision").decision();
```

  - You can also specify a list of trigger names to be OR-ed explicitly:

```
m_tisTos->setOfflineInput();
m_tisTos->setTriggerInput(); // reset trigger names
m_tisTos->addToTriggerInput("Hlt1DiHadronDecision");
m_tisTos->addToTriggerInput("Hlt1MuTrack4JPsiDecision");
bool hlt1MixDec= m_tisTos->triggerTisTos().decision();
```

  - Passing a vector of names also works:

```
std::vector<std::string> mx;
mx.push_back("Hlt1DiHadronDecision");mx.push_back("Hlt1*Mu*Decision");
m_tisTos->setTriggerInput(mx);
bool hlt1Mix2Dec= m_tisTos->triggerTisTos().decision();
```

Accessing decisions via TriggerTisTos tool

- Python version:

```
tsvc= appMgr.toolsvc()  # get TisTos tool right before appMgr.run()
m_tisTos= tsvc.create('TriggerTisTos',interface='ITriggerTisTos')
          # after processing an event i.e. appMgr.run(1)
m_tisTos.setOfflineInput() # dummy signal
hlt1MuDec = m_tisTos.triggerTisTos('Hlt1*Mu*Decision').decision() # 0/1
```

– You can also specify a list of trigger names to be OR-ed explicitly:

```
m_tisTos.setOfflineInput()
m_tisTos.setTriggerInput()
m_tisTos.addToTriggerInput('Hlt1DiHadronDecision')
m_tisTos.addToTriggerInput('Hlt1MuTrack4JPsiDecision')
hlt1MixDec= m_tisTos.triggerTisTos().decision()
```

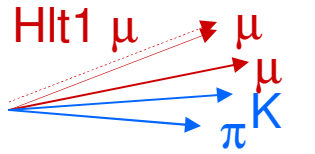
python

– Passing a vector of names also works:

```
m_tisTos.setTriggerInput() # reset trigger input
mx = m_tisTos.triggerSelectionNames() #trick: empty vector<string>
mx.push_back('Hlt1DiHadronDecision')
mx.push_back('Hlt1*Mu*Decision')
m_tisTos.setTriggerInput(mx) # define trigger input
hlt1Mix2Dec= m_tisTos.triggerTisTos().decision()
```

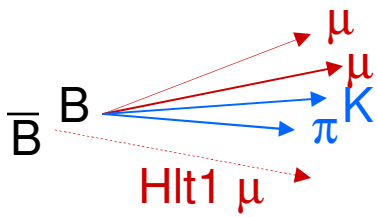
TIS and TOS classifications

- Signal – off-line selected particle (simple or composite) e.g. $B_d \to \mu\mu K^*$, $K^* \to K\pi$ candidate
- Can classify each trigger decision (individual trigger line or OR between many lines) as:
  - **TOS** – **T**rigger **O**n **S**ignal
  - **TIS** – **T**rigger **I**ndependent of **S**ignal

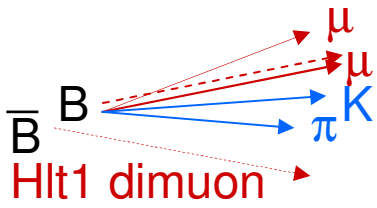


TOS

- Four types of trigger decisions:
  - Neither TOS nor TIS (called "TOB")
  - TOS but not TIS
  - TIS but not TOS
  - TOS and TIS at the same time



TIS



TOB

- Classification useful for:
  - Trigger optimization (e.g. TOB events are undesired)
  - Trigger monitoring
  - Determination of trigger efficiencies and trigger biases in physics analysis from data themselves

Classified trigger decisions

(see slide 22 for L0 decisions!)

C++

```
Particle signal= ...your_offline_signal_candidate;
ITriggerTisTos::TisTosDecision classifiedHlt2Dec =
                    m_tisTos->triggerTisTos(signal, "Hlt2*Decision" );
info() << " Hlt2 decision= " << classifiedHlt2Dec.decision()
       << " TIS= "  << classifiedHlt2Dec.tis()
       << " TOS= "  << classifiedHlt2Dec.tos()
       << " TOB= "  << ( classifiedHlt2Dec.decision() &&
      !classifiedHlt2Dec.tos() && !classifiedHlt2Dec.tis() ) << endmsg;
```

python

```
signal= evt['/Event/Phys/PreselB2Charged2Body/Particles'][0]
classifiedHlt2Dec = m_tisTos.triggerTisTos(signal,'Hlt2*Decision')
print ' Hlt2 decision= ', classifiedHlt2Dec.decision(),
print ' TIS= ', classifiedHlt2Dec.tis(),
print ' TOS= ', classifiedHlt2Dec.tos(),
print ' TOB= ', int(classifiedHlt2Dec.decision() and \
           not classifiedHlt2Dec.tos() and not classifiedHlt2Dec.tis())
```

- `decision()` does not depend on the signal !
- The Trigger Input [here `"Hlt2*Decision"`] can be specified in other ways shown on slide 9
- Also the Offline Input [here `Particle signal`] can be specified ahead of the `m_tisTos->triggerTisTos` call,
  at once by `setOfflineInput(signal)`
  or part by part using `addToOfflineInput(part_of_the_signal)`.

Classified trigger decisions

- You can also obtain a list of triggers (within the scope of the specified Trigger Input) which match a certain TIS/TOS classification:

```
// after the Offline and Trigger Inputs have been defined, call:

std::vector<std::string> triggerSelectionNames(
        unsigned int decisionRequirement = ITriggerTisTos::kAnything,
        unsigned int tisRequirement = ITriggerTisTos::kAnything,
        unsigned int tosRequirement = ITriggerTisTos::kAnything )
```

C++

```
// getting exclusive-TOS (i.e. TOS and not TIS) Hlt1 triggers:
std::vector<std::string> exclTOSHlt1Triggers =
          m_tisTos->triggerSelectionNames(signal,
                                          "Hlt1*Decision“,
                        ITriggerTisTos::kTrueRequired,
       ITriggerTisTos::kFalseRequired, ITriggerTisTos::kTrueRequired);


// all known Hlt2 trigger lines
//                            (no Tis-Tos-Tobbing here, don’t need signal)
std::vector<std::string> hlt2Triggers =
                       m_tisTos->triggerSelectionNames("Hlt2*Decision");
```

Classified trigger decisions

- Python version:

python

```
# getting exclusive-TOS (i.e. TOS and not TIS) Hlt1 triggers:
exclTOSHlt1Triggers = \
         m_tisTos.triggerSelectionNames(signal,\
                                        'Hlt1*Decision',\
                        m_tisTos.kTrueRequired,\
                m_tisTos.kFalseRequired,m_tisTos.kTrueRequired)
# all known Hlt2 trigger lines
#                             (no Tis-Tos-Tobbing here, don't need signal)

hlt2Triggers = m_tisTos.triggerSelectionNames('Hlt2*Decision')
```

Access to candidates selected by triggers

- `HltSelReports` have information about selected candidates (Tracks, Vertices, Particles, …)
- I don't expect many users will want to access `HltSelReports` directly
  - if you do, this container has analogous structure to `HltDecReports` (see its doxygen documentation)
- I recommend using `TriggerTisTos` tool instead

C++

```
// hlt summaries of Particles passing Hlt2UnbiasedDiMuonDecision
std::vector<const LHCb::HltObjectSummary*> selectedTriggerCandidates=
          m_tisTos->hltObjectSummaries("Hlt2UnbiasedDiMuonDecision");

// the following will also work, but you may get
//                          a mix of summaries of Tracks and RecVertecies
selectedTriggerCandidates=
          m_tisTos->hltObjectSummaries("Hlt1*Decision");
```

python

```
selectedTriggerCandidates = \
m_tisTos.hltObjectSummaries('Hlt2UnbiasedDiMuonDecision')

selectedTriggerCandidates = m_tisTos.hltObjectSummaries('Hlt1*Decision')
```

Using TisTos tool to classify Trigger selection candidates

- You can obtain a list of selected candidates which are classified as TOS, TIS, TOB (or combination of them)

```
// after the Offline and Trigger Inputs have been defined, call:

std::vector<const LHCb::HltObjectSummary*> hltObjectSummaries(
        unsigned int tisRequirement = ITriggerTisTos::kAnything,
        unsigned int tosRequirement = ITriggerTisTos::kAnything )
```

Tis Requirement	Tos Requirement	Result
kTrueRequired	kTrueRequired	cannot be satisfied
kFalseRequired	kFalseRequired	TOB trigger objects (i.e. neither TIS nor TOS);
kFalseRequired	kTrueRequired	TOS trigger objects (same as kAnything,kTrueRequired)
kTrueRequired	kFalseRequired	TIS trigger objects (same as kTrueRequired,kAnything)
kFalseRequired	kAnything	TOS and TOB trigger objects
kAnything	kFalseRequired	TIS and TOB trigger objects
kAnything	kAnything	all trigger objects

For trigger candidates TIS, TOS, TOB are all mutually exclusive (unlike for decisions!)

List of classified trigger selection candidates

- Code example:

C++

```
// getting TOS objects
std::vector<const LHCb::HltObjectSummary*> tosCandidates =
          m_tisTos->hltObjectSummaries(signal,
                                       "Hlt1SingleMuonIPCLODecision“,
       ITriggerTisTos::kFalseRequired, ITriggerTisTos::kTrueRequired);
// .. and now TOB objects (signal and Trigger Input already defined)
std::vector<const LHCb::HltObjectSummary*> tobCandidates =
m_tisTos->hltObjectSummaries(
    ITriggerTisTos::kFalseRequired, ITriggerTisTos::kFalseRequired);



// check if the signal was used in the Hlt1 in any way (TOS or TOB):
if( m_tisTos->hltObjectSummaries(signal, "Hlt1*Decision",
ITriggerTisTos::kFalseRequired, ITriggerTisTos::kAnything ).size() ){
        info() << "The signal was used." << endmsg; }
```

List of classified trigger selection candidates

python

```
# getting TOS objects
tosCandidates = m_tisTos.hltObjectSummaries(signal,\
                        'Hlt1SingleMuonIPCLODecision',\
                        m_tisTos.kFalseRequired, m_tisTos.kTrueRequired)
# .. and now TOB objects (signal and Trigger Input already defined)
tobCandidates = m_tisTos.hltObjectSummaries(\
                        m_tisTos.kFalseRequired, m_tisTos.kFalseRequired)



# check if the signal was used in the Hlt1 in any way (TOS or TOB):
if m_tisTos.hltObjectSummaries(signal, 'Hlt1*Decision',\
              m_tisTos.kFalseRequired, m_tisTos.kAnything ).size():
                        print 'This signal was used'
```

HltObjectSummary class

- Since selected candidates cannot be saved with full info they are summarized
- Same class for all types of candidates (Particle,RecVertex,Track, anything we add in the future):
  - Vector containing either pointers to substructure or LHCbIDs
  - Numerical information with a string key identifying type of information:
    - Standard Info: same set of variables for each object of the same class (has a "#" in the label)
    - Extra Info: each Hlt1 filter adds one (history of Hlt1 processing)
  - Class ID of summarized class (also pointer to the object itself; on-line use only)
- See my talk on Hlt Raw Data at Dec.08 Software Week for more information

HltSelReportsMaker VERBOSE key 3{ summarizedObjectCLID : 10010 numericalInfo : { 0#Track.firstState.z:-53.5628 , 1#Track.firstState.x:-0.0431684 , 2#Track.firstState.y:-0.00109219 , 3#Track.firstState.tx:0.00133175 , 4#Track.firstState.ty:-0.0526368 , 5#Track.firstState.qOverP:-2.85726e-05 , IP_PV2D:0.040732 , IsMuon:1 , L0ConfirmWithT:0 , PT:1840.25 , PT0:3574.56 , RZVeloTMatch_Hlt1SingleMuonPrepTFTConf:-47.6819 , Tf::PatVeloSpaceTool:38 , chi2_PatMatch:0.161546 , } substructure : { } lhcbIDs : { 269033390 , 269031334 , 269027662 , 269025601 , 269024971 , 269022912 , 269019314 , 269017249 , …, } }

HltSelReportsMaker VERBOSE key 4{ summarizedObjectCLID : 10030 numericalInfo : { 0#RecVertex.position.x:-0.0460179 , 1#RecVertex.position.y:0.0697576 , 2#RecVertex.position.z:-54.4946 , DOCA:0.0436636 , VertexDimuonMass:5270.4 , } substructure : { 3:10010 , 0:10010 , } lhcbIDs : { } }

Converting object summaries to object themselves

- In on-line environment you can go from the HltObjectSummary to the Object itself

C++

```
// templated function to convert object summary to object itself
 template <class T>
 const T* hltObject(const LHCb::HltObjectSummary* hos )
 {
     if( !(hos->summarizedObject()) )
        return (const T*)(0); // means pointer to the object is not available
     if( hos->summarizedObjectCLID()
             != T::classID()  )
        return (const T*)(0); // means different object than requested
     return dynamic_cast<const T*>( hos.summarizedObject() ) ); // convert
 }

 // get a Track from the Track summary:
  const LHCb::HltObjectSummary* hos = …;
  const LHCb::Track* triggerTrack = hltObject<LHCb::Track>( hos );
```

- No need to cast in python! An object returned by the summarizedObject() method is self-aware.

python

```
triggerTrack = hos.summarizedObject()
```

Classifying L0 decisions via Hlt1L0 trigger lines

- You can get L0 trigger decisions from L0DUReport
- Each L0 trigger (XXX=Muon, 'Muon,lowMult', MuonNoGlob, DiMuon, 'DiMuon,lowMult', Hadron, Electron, Photon, LocalPi0, GlobalPi0) has a corresponding Hlt1 pass-through line - Hlt1L0XXXDecision
  - True Hlt1 trigger lines. If decision()==true then Hlt1Global decision() must be true as well.
  - **Heavily postscaled:**
    - At present one in a million kept
    - They convert candidates selected by L0 triggers to Hlt1 type selections
    - Postscaling means that they are allowed to fully execute i.e. create candidates if the L0 XXX line succeeded, but the decision turned to false most of the time anyway

See slides 5 and 6:

decisionName : Hlt1L0DiMuon,lowMultDecision HltDecReport : { decision : 0 errorBits : 0 numberOfCandidates : 0 executionStage : 2 intDecisionID : 14 }
decisionName : Hlt1L0DiMuonDecision HltDecReport : { decision : 0 errorBits : 0 numberOfCandidates : 2 executionStage : 6 intDecisionID : 12 }

L0 DiMuon,lowMult decision was false

L0 DiMuon decision was true

- Classifying Hlt1L0XXXDecision decision the normal way (slide 12) not useful since decision()==false essentially all the time
- There is a work around (see next slide)

Classifying L0 decisions

- Compare with slide 12. The differences are highlighted in red.

C++

```
Particle signal= ...your_offline_signal_candidate;
m_tisTos->setOfflineInput(signal);
ITriggerTisTos::TisTosDecision classifiedL0Dec =
m_tisTos->selectionTisTos(
                  m_tisTos->triggerSelectionNames("Hlt1L0*Decision") );
bool l0Dec = m_tisTos->hltObjectSummaries("Hlt1L0*Decision").size()!=0;
info() << " L0 decision= " << l0Dec
       << " TIS= "  << classifiedL0Dec.tis()
       << " TOS= "  << classifiedL0Dec.tos()
       << " TOB= "  << ( l0Dec &&
      !classifiedL0Dec.tos() && !classifiedL0Dec.tis() ) << endmsg;
```

python

```
m_tisTos.setOfflineInput(signal)
classifiedL0Dec = m_tisTos.selectionTisTos(  \
                  m_tisTos.triggerSelectionNames('Hlt1L0*Decision') )

l0Dec = m_tisTos.hltObjectSummaries('Hlt1L0*Decision').size()!=0
print ' L0 decision= ', int(l0Dec),
print ' TIS= ', classifiedL0Dec.tis(),
print ' TOS= ', classifiedL0Dec.tos(),
print ' TOB= ', int( l0Dec and \
    not classifiedL0Dec.tos() and not classifiedL0Dec.tis() )
```

For Hlt1 developers

- By default only selected candidates by the Decision step of each trigger line are saved
- If you are interested in debugging intermediate Hlt1 selections, you can add them to HltSelReports by changing configuration of its Maker

```
from Configurables import HltSelReportsMaker
HltSelReportsMaker().DebugEventPeriod = 1
#   or
HltSelReportsMaker().MaxCandidatesNonDecision =500
```

- You can then use TriggerTisTos tool to access (and classify) their decisions and selected candidates

HltVertexReports

- To allow easy access to vertex info (primary vertices, Velo vertices) without overhead of saving all contributing Tracks (unlike HltSelReports)
- Container of SmartRefVector<LHCb::VertexBase> keyed with the name of the trigger algorithm making RecVertices (e.g. "PV2D")
  - Saved vertices can also be accessed as VertexBase::Container at /Event/Hlt/VertexReports/PV2D/
  - Covariance matrix not stored

C++

```
#include "Event/HltVertexReports.h"
HltVertexReports* vtxReports =
                        get<HltVertexReports>(HltVertexReportsLocation::Default);

// get HltVertexReport for PV2D
const HltVertexReports::HltVertexReport* vtxRep = vtxReports->vertexReport("PV2D");
if( vtxRep ){
for( HltVertexReports::HltVertexReport::const_iterator iv=vtxRep->begin();
                                                    iv!=vtxRep->end(); ++iv){
 const VertexBase & v = **iv;
 info()<<" x "<<v.position().x()<<" y "<<v.position().y()<<" z "<< v.position().z()
       <<" chi2 " << v.chi2()<< " ndf " << v.nDoF()<< endmsg;
} }
```

HltVertexReports

- Python version:

python

```
vtxReports = evt['/Event/Hlt/VertexReports']

# get HltVertexReport for PV2D
vtxRep = vtxReports.vertexReport('PV2D')
if vtxRep!=None:
    for iv in vtxRep:
        v = iv.target()
        print ' x ',v.position().x(),' y ',v.position().y(),' z ',v.position().z(),
        print ' chi2 ',v.chi2(),' ndf ',v.nDoF()
```

More documentation

- Twiki TriggerTisTos tool page at https://twiki.cern.ch/twiki/bin/view/LHCb/TriggerTisTos
  - Includes up-to-date extensive python sample code
  - Links to previous presentations
  - Comments for each release (please switch to DaVinci v23r1 !)