HLT2 line development

Marian Stahl

University of Cincinnati

LHCb Starterkit-For-All Full Run 3 Edition

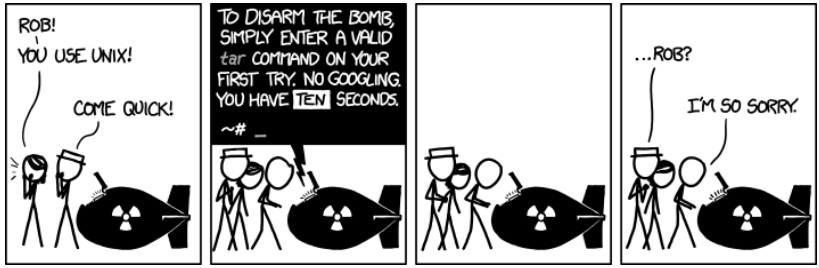
March 15, 2022

Outline

- The session will be split in two parts:
  1. will be following the MOORE tutorial, and you are invited to follow the steps interactively.
  2. will be a hands-on part, where you write your own $B_s^0 \longrightarrow J/\psi\phi$ "persist reco" and/or "selective persistence" lines.
     You will run your lines on B_c^+ signal MC and use the output in the following sessions.

- If you have not extracted the stack tarball yet, please do.

```
tar -xzf /eos/lhcb/user/a/ascarabo/stack_starterkitRun3.tar.gz -C .; cd stack
```

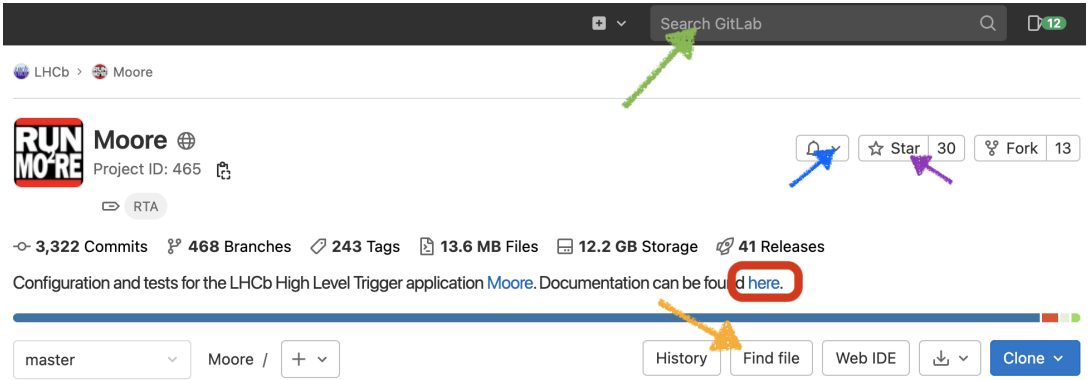


- If you feel comfortable, you can of course use your private stack.

  It needs to be up to date with master from yesterday noon CERN time.

- Another option is `lb-run -nightly lhcb-master Moore/master gaudirun.py` and a local checkout of Moore `git clone ssh://git@gitlab.cern.ch:7999/lhcb/Moore.git` or `git clone https://gitlab.cern.ch/lhcb/Moore.git`

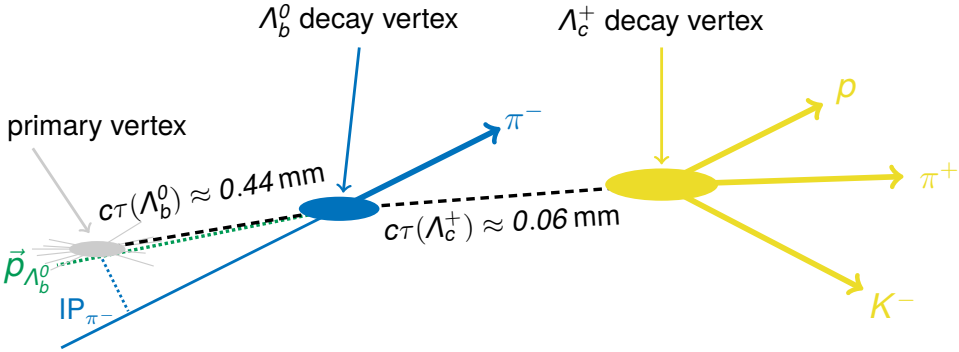
- The MOORE gitlab page is a good starting point.



- Direct links to gitlab, docs, Mattermost, Twiki, Mailing list.

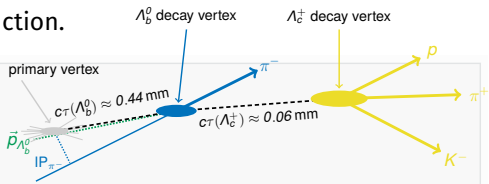
Hint: Also look at the Twiki page of your WG. There could be more tips and instructions.

- MOORE is a repository of files used to configure a GAUDI application.
  Most files define application configurations for HLT1 or HLT2.
- An HLT line is a sequence of steps that collectively define whether an event contains an object of interest.
- Our object of interest are candidates of the decay $\Lambda_b^0 \to \Lambda_c^+ \pi^-$;
  we later add $\Lambda_b^0 \to \Lambda_c^+ \mu^- \overline{\nu}_\mu$ to highlight certain parts of the configuration.



- We start by sketching the line defining function.

```python
def lb0_to_lcpim_line():
    pvs = make_pvs()
    protons = protons_for_charm()
    kaons = kaons_for_charm()
    pions = pions_for_charm_and_beauty()
    lcs = make_lambdacs(protons, kaons, pions, pvs)
    lbs = make_lambdabs(lcs, pions, pvs)

    return lbs
```



- Is this what you expected? Can you think of something different?
- Each step will run one, or a sequence of C++ algorithms from the LHCb software stack.
- We now need to adapt it to Moore.

- On PVs: In Run 2, PVs were handled implicitly. In Run 3, they are explicit in the selection config.

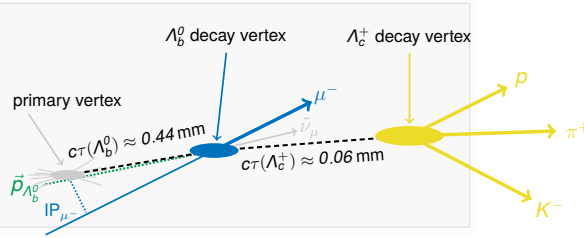
- Reminder: An HLT line is a sequence of steps [...] defining whether an event contains an object of interest.
  Our function has to return an object that encodes this: namely `Moore.lines.Hlt2Line`.

```python
from Moore.lines import Hlt2Line

def lb0_to_lcpim_line(name="Hlt2Tutorial_Lb0ToLcpPim_LcpToPpKmPip_Line", prescale=1):
    pvs = make_pvs()
    protons = protons_for_charm()
    kaons = kaons_for_charm()
    pions = pions_for_charm_and_beauty()
    lcs = make_lambdacs(protons, kaons, pions, pvs)
    lbs = make_lambdabs(lcs, pions, pvs)

    return Hlt2Line(
        name=name,
        algs=[lbs],
        prescale=prescale,
     )
```

- We have given the line a name, and an optional prescale.
- The `algs` parameter defines the sequence of steps: the **control flow**. It evaluates whether this line made a positive *decision* or not.

- Note that we did not have to pass any of the other objects to `algs`.
  MOORE can automatically deduce what algorithms need to be run to get `lbs` in our case.
  This is known as **data flow**, defining which inputs are necessary to create a given output.
- The **control flow** on the other hand may be configured by you as a line author.
  In the default setup, the list of objects passed to `algs` defines the *order* in which algorithms are executed.
- In the example snippet, we require that the sequence of algorithms to reconstruct muons only runs if there is a reconstructed PV in the event; and only if there was a positive muon decision, all other steps are executed to build our Λ_b^0.

```python
from RecoConf.hlt1_tracking import require_pvs


    return Hlt2Line(
        name=name,
        algs=[require_pvs(pvs), muons, lbs],
        prescale=prescale,
    )
```



- The control flow lets us add additional algorithms, such as monitors.
- There are two ways to monitor HLT2 output:
  1) Online (directly during data taking using histograms):
     - distributions of signal candidates (mass, PT, ETA, phi) will be produced automatically, like in Run 2.
     - adding customised distributions is also possible, but exact configuration is still under discussion
     - distributions will be monitored by HLT piquet/data manager
  2) Offline (Analysis Productions):
     - more elaborate monitoring possible, larger datasets, analysis on ntuples
     - complementing the online monitoring (can also provide cross-checks for online monitoring)
     - monitored by WGs
     - see for more info Dylan's presentation on Friday and this tutorial
  - Focus now on offline checks from AP.

- Our line-defining function is almost ready. There are two more things to consider:
  - A bookkeeping step to *register* the line centrally.
  - Adding a list of reconstruction objects to the control-flow.

```
@register_line_builder(all_lines)
def lb0_to_lcpim_line(name="Hlt2Tutorial_Lb0ToLcpPim_LcpToPpKmPip_Line", prescale=1):


    return Hlt2Line(
        name=name,
        algs=upfront_reconstruction() + [require_pvs(pvs), lbs],
        prescale=prescale,
     )
```

- We now need to define how to actually reconstruct and select p, K, π, Λ_c^+ and Λ_b^0.
- The `standard_particles` module contains various types of *proto-particles* and a few generic combined particles.
- For our example, we need long tracks with RICH information and p, K, π mass hypotheses.
- We also need PVs, as they are explicit reconstruction objects.

```
from ..standard_particles import (
    make_has_rich_long_kaons,
    make_has_rich_long_pions,
    make_has_rich_long_protons,
)
from RecoConf.reconstruction_objects import make_pvs_v2 as make_pvs
```

- We use ThOr functors to filter candidates.

```
from GaudiKernel.SystemOfUnits import GeV
import Functors as F

from ..algorithms_thor import ParticleFilter, require_all


def protons_for_charm():
    pvs = make_pvs()
    cut = require_all(
        F.PT > 0.5 * GeV,
        F.MINIPCHI2(pvs) > 9.,
        F.PID_P > 5.,
    )
    return ParticleFilter(make_has_rich_long_protons(), F.FILTER(cut))
```

- The function has no arguments. This is on purpose for *production-ready* selections. In this way we have re-defined a custom basic building block for our lines.

Attention

Every call to `ParticleFilter` or `ParticleCombiner` with different inputs or different selection cuts will create a new instance of the algorithm.

- We now combine our basic particles to reconstruct a Λ_c^+ decay.
- There is detailed documentation for combiners in our codebase. Most notably:
  - The order of particles in the decay descriptor and the input list must be the same; there is no mix and match unlike Run 2!
  - Another change w.r.t. Run 2 is that particles of the same type are passed explicitly `([pi, pi, pi], DecayDescriptor="[D+ -> pi+ pi+ pi-]cc")`
  - Multiple child particles with the same ID must be grouped together (`D+ -> pi+ pi+ pi-` is good, `D+ -> pi+ pi- pi+` is forbidden).
  - For performance purposes, the algorithm logic assumes that the rarest children are listed first in the decay descriptor. In case you are unsure what is rarest, checking counters in the log file can help.
- Combiners (algorithms in general) with the exact same inputs and configuration are de-duplicated. What follows is again:

Attention

Every call to `ParticleFilter` or `ParticleCombiner` with different inputs or different selection cuts will create a new instance of the algorithm.

```python
from Functors.math import in_range
from GaudiKernel.SystemOfUnits import (GeV, MeV, mm)

from ..algorithms_thor import ParticleCombiner

def make_lambdacs_for_beauty(protons, kaons, pions, pvs):
    two_body_combination_code = require_all(
        F.MAXDOCACHI2CUT(9.), F.MAXDOCACUT(0.1 * mm))
    combination_code = require_all(
        in_range(2080 * MeV, F.MASS, 2480 * MeV),  # mass of the combination
        F.PT > 1.4 * GeV,  # pT of the 3-track combination
        F.SUM(F.PT) > 2 * GeV,
        F.MAXDOCACHI2CUT(9.),
        F.MAXDOCACUT(0.1 * mm),
    )
    vertex_code = require_all(
        in_range(2100 * MeV, F.MASS, 2460 * MeV),  # mass after the vertex fit
        F.PT > 1.6 * GeV,  # pT after the vertex fit
        F.CHI2DOF < 10.,
        F.BPVFDCHI2(pvs) > 25.,
    )
    return ParticleCombiner(
        [protons, kaons, pions],
        DecayDescriptor="[Lambda_c+ -> p+ K- pi+]cc",
        name="Tutorial_Lcp_Combiner",
        Combination12Cut=two_body_combination_code,
        CombinationCut=combination_code,
        CompositeCut=vertex_code,
    )
```

- When developing a line, you may want to test several different selections. You could
  - edit the source file; or
  - use the `@configurable` decorator to configure multiple instances of your line from the run-script and run all of them in parallel.
- This is demonstrated in the full example.

⟷ Let's take a look at the example scripts, run them and inspect the output.

Hint

The full example runs 3 lines. One for the $\Lambda_b^0 \to \Lambda_c^+ \pi^-$ decay, and two instances of the $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ line with a slightly modified pion p_{T} cut. Have a close look at the counters that the example produces. Can you understand all of them?

- We have two counters for the `Tutorial_pions_for_charm_and_beauty` filter. Which one is which?
- Why does one of them have fewer inputs?
- We have two counters for the `Tutorial_Lcp_Combiner` and three for `Tutorial_Lb0_Combiner` combiners, two of which look identical. Why is that?
- Some of the combiners don't seem to run on all 100 events. Why is that?
- There is no `Combination12Cut` counter for `Tutorial_Lcp_Combiner#1`, even though the input containers for one event are not empty. What happened?

- We have learned how to write and test-run an HLT2 line.
  Full documentation of this tutorial
  Also take note of the best practices for writing HLT2 lines.

- Now it's time for the hands-on part!

- You want to study secondary $B_s^0 \longrightarrow J/\psi\phi$ decays in Run 3 and have signal MC for $B_c^+ \longrightarrow B_s^0\pi^+$ and $B_c^+ \longrightarrow B_s^0\mu^+\nu_\mu$.
- But there might be more than these two decays, and you decide to write either a "persist reco" or "selective persistence" line, or study both initially.
- For quick studies, you can run on a minimum bias sample. When you have finished the prototype, you can produce your own `dst` file running on one of the two signal MC samples.
- You can then use this `dst` file for the following sessions on Sprucing, DaVinci and the HltEfficiencyChecker.

[JINST 14 P04006]

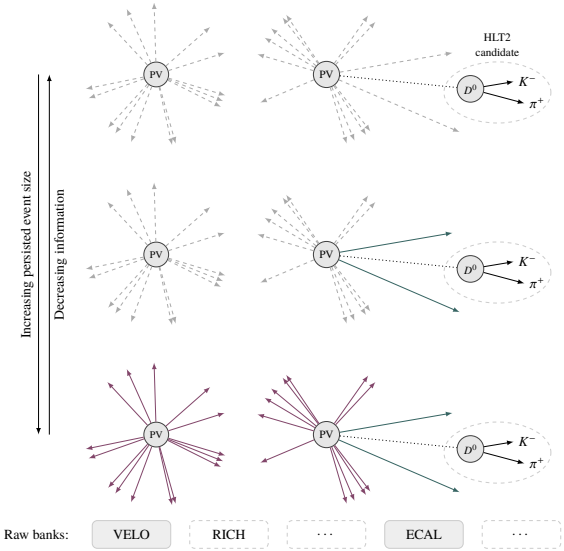
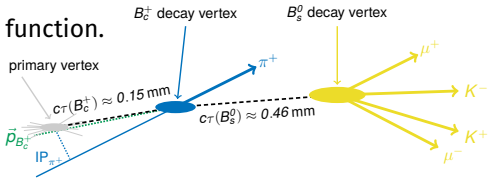


Figure 3. A cartoon of the same reconstructed event with varying levels of object persistence: Turbo (top); selective persistence (middle); and complete reconstruction persistence (bottom). Solid objects are those persisted in each case. A trigger selection may also ask for one or more sub-detector raw banks to also be stored, shown as solid rectangles.

- You can copy the code from the tutorial as a starting point.
  - Where would you put your line in the Moore repository?
- It should be easiest to start from the line-defining function.
  - Which objects do you need?
  - Is $B_s^0 \longrightarrow J/\psi\phi$ a two- or four-body decay? Does it make a difference?
  - How to configure the control flow?
  - How to switch on persist reco or configure selective persistence?



- Would you want to use shared objects other than protoparticles? Why (not)?
- Coming up with good variables and cut values for your selection can be difficult. You may want to browse through Moore, or discuss with others.
- You may find what looks like a solution; but beware of badly chosen cut values!

- The run script is in `Hlt/Hlt2Conf/options/run_starterkit_bs_to_jpsiphi.py`
- You need to modify it to pick up your line
- The script only includes $B_c^+ \longrightarrow B_s^0 \mu^+ \nu_\mu$ signal MC. To run the hadronic mode, use

```
options.input_files = [
    f"root://eoslhcb.cern.ch//eos/lhcb/grid/prod/lhcb/MC/Upgrade/XDIGI/00143675/0000/00143675_000000{i}_1.xdigi"
    for i in [
        "15", "26", "37", "34", "41", "44", "47", "43", "46", "49",
        "45", "51", "55", "56", "57", "59", "61", "63", "65", "66",
        "70", "76", "79", "81", "82", "85", "88", "91", "96", "97",
        "98", "99", "64", "23", "95", "33", "84", "48", "80", "87",
        "54"
    ]
]
```

- Compare your solution to `Hlt/Hlt2Conf/python/Hlt2Conf/lines/starterkit/bs_to_jpsiphi.py`.
  - What are the differences?
  - How did you answer the questions 2 slides ago? How did they do it?

- We hope you enjoyed this session and will enjoy the following ones, teaching you what you can do with the data and how to find a suitable selection for your line.
- We are looking forward to many merge requests with new or updated HLT2 lines!