Table of Contents

Table of Contents

PIDCalib Packages

This page provides information on how to use the PIDCalib package for extracting PID performance results from both collision data and MC. ***Questions or comments concerning the code may be sent to the following mailing list***: lhcb-phys-pid-calibration@cern.ch. Please use ***this*** mailing list for any question that involves PIDCalib. More general information on PID can be found in the Charged PID twiki.

Latest PIDCalib setup instructions

Urania has recently been migrated to git, and the following instructions for setting up PIDCalib are thus git oriented. In order to access the new Run 2 Working Group Production (WGP) calibration nTuples, and the previous Run 1 samples, please follow these setup instructions:

```
lb-dev Urania/v8r0

cd UraniaDev_v8r0

git lb-use Urania

git lb-checkout Urania/master PIDCalib/PIDPerfTools

git lb-checkout Urania/master PIDCalib/PIDPerfScripts

git checkout Urania/v8r0 cmake

make configure

make install
```

In order to access the calibration samples via EOS, one must have a valid grid proxy. Before running PIDCalib, please obtain a fresh proxy:

```
lhcb-proxy-init
```

Finally, in order to run any PIDCalib scripts, a bash shell must be loaded:

```
./run bash --norc
```

PIDCalib samples for Run 2 data available

The PID calibration samples for Run 2 have been produced via Working Group Production (WGP). These samples are selected within TurCal, and are stored in nTuple format. **Electrons in 2015 and 2016 are an exception to this, and have been selected and processed outside of the WGP. In 2015, the electrons were selected by TurCal (online and offline) and produced outside of the WGP, but in 2016 due to the electron reconstruction bugs, the electron samples have been produced via Stripping 28 (offline). In 2017 and 2018, the electrons were selected by TurCal and processed using the WGP, but additional post-WGP work was done to correctly compute the sWeights.**

To access the samples within PIDCalib, first follow the setup instructions above. To make the performance histograms, execute the following script from within UraniaDev _v8r0:

```
python PIDCalib/PIDPerfScripts/scripts/python/MultiTrack/MakePerfHistsRunRange.py
```

You will see a help menu with suggestions for which arguments to provide. In particular, the argument called **sample version** is used to select the data you wish to run on.

For **2015** data, use **"Turbo15"** for sample version. *This does not apply to 2015 electrons.*

For **2016** data, use **"Turbo16"** for sample version. *This does not apply to 2016 electrons.*

For **2017** data, use **"Turbo17"** for sample version. *This does not apply to 2017 electrons.*

For **2018** data, use **"Turbo18"** for sample version. *This does not apply to 2018 electrons.*

For **2015** electrons, use **"Electron15"** for sample version. *This is a TurCal sample produced outside the WGP.*

For **2016** electrons, use **"Electron16"** for sample version. *This is a Stripping 28 sample produced outside the WGP.*

For **2017** electrons, use **"Electron17"** for sample version. *This is a TurCal sample produced with the WGP, but with additional offline-computed sWeights.*

For **2018** electrons, use **"Electron18"** for sample version. *This is a TurCal sample produced with the WGP, but with additional offline-computed sWeights.*

For **2016 pA** data, use **"pATurbo16"** for sample version.

For **2016 Ap** data, use **"ApTurbo16"** for sample version.

The Run 1 samples are also available, and can be accessed by providing a stripping version e.g. "20".

The table below summarises which sample versions correspond to each year of data taking.

Sample Version	Comment
Turbo15	Selects the pp 2015 Run 2 turbo data from WGP (excluding electrons)
Turbo16	Selects the pp 2016 Run 2 turbo data from WGP (excluding electrons)
Turbo17	Selects the pp 2017 Run 2 turbo data from WGP (excluding electrons)
Turbo18	Selects the pp 2018 Run 2 turbo data from WGP (excluding electrons)
pATurbo16	Selects the pA 2016 Run 2 turbo data from WGP
Electron15	Selects the pp 2015 Run 2 turbo electron data, produced outside WGP
Electron16	Selects the pp 2016 Run 2 Stripping 28 electron data, produced outside WGP
Electron17	Selects the pp 2017 Run 2 turbo electron data, produced with WGP + additional offline sWeights
Electron18	Selects the pp 2018 Run 2 turbo electron data, produced with WGP + additional offline sWeights
ApTurbo16	Selects the Ap 2016 Run 2 turbo data from WGP
20	Selects Stripping20 for pp 2012 data
20r1	Selects Stripping20r1 for pp 2011 data

Which variables are available within PIDCalib?

A full list of the available variables can be viewed within the script TupleDataset.py:

```
cat PIDCalib/PIDPerfScripts/python/PIDPerfScripts/TupleDataset.py
```

A range of PID variables are defined within the array called **vars_dataset**, along with various other variables which can either be cut on or used to bin the sample. By default, no cuts are placed on the sample.

The binning variables are selected on the command line, and cuts can also placed on the data (using the -c option). If you wish to use a non-standard binning scheme, you will need to provide your own scheme in a separate script. Instructions for how to do this are given below

When declaring new binnings, please respect the variable names as defined in TupleDataset.py. Also, when selecting your PID variables, binning variables and cuts, please ensure that all variable names match those defined in TupleDataset.py

If you have added a bin shceme to PIDCalib, or modified your local copy of the code in any other way, you must recompile before running again:

```
cd UraniaDev _v8r0

make purge

make

./run bash
```

Which should I use?

For the Run 2 samples, only the "MC15TuneV1" and "MC12TuneV4" ProbNN PID variables should be used. The "MC12TuneV2" and "MC12TuneV3" variables were found to have suboptimal performance for 2016 data. For all Run 2 analyses, it is recommended to use the "MC15TuneV1" ProbNN variables - those of "MC12TuneV4" are optional.

For the Run 1 samples, only "MC12TuneV2" and "MC12TuneV3" are accessible.

Variable name	Comment
MC15TuneV1_ProbNN	Recommended for Run 2 (**Default since DaVinci version v40r0**)
MC12TuneV2_ProbNN	Deprecated and not available for Run 2 samples (**Default before and after Da Vinci version v40r0**)
MC12TuneV3_ProbNN	Deprecated and not available for Run 2 samples
MC12TuneV4_ProbNN	Sub-optimal for Run 2

The different tunings can be added to your ntuples using TupleToolANNPID. If you use the TupleToolPID only the default version will be stored.

If your Turbo or stripping lines already applied ProbNN cuts, then a default version was running in the production. The correspondence between the versions and the tunings is

Sample	ProbNN tuning	Comment
S20r1, S20	MC12TuneV2 _ProbNN	

S21r1, S21r1p1, S21, S21r0p1	MC12TuneV2 _ProbNN	
S22, S22r0p1, S23, S24, S24r0p1	MC12TuneV2 _ProbNN	2015 data with Run 1 tuning
S26, S28	MC15TuneV1 _ProbNN	
Turbo 02a, 03, 03pLead	MC15TuneV1 _ProbNN	

Note that ProbNN variables are available in Turbo since 2016 data taking, so no ProbNN cuts are applied to Turbo02 (2015) data.

See this presentation (slide 10) for the description of main differences between Run 1 and Run 2 ProbNN tunes.

Important information for users of proton

The ProbNN variables are correlated with track displacement (e.g. MINIPCHI2), which can cause problems for analysts if a PID calibration sample has different lifetime than your signal. As the default proton calibration sample is long-lived (Lambda -> p), the effect becomes apparent for ProbNNp.

The issue is documented in a presentation from Anton:

https://indico.cern.ch/event/668111/contributions/2731718/attachments/1530806/2395860/probnnp.pdf

An alternative proton calibration sample "P_IncLc" can be used in PIDCalib to determine ProbNNp efficiencies, and should be compared with the default "P" sample. At present, the "P_IncLc" sample is only available for Run 1 data, but work is ongoing to include this sample for Run 2.

Yandex PID variables (yPID)

The Working Group Production samples also contain a set of MVA PID variables provided by Yandex, which are trained using the same inputs as ProbNN but use different algorithms. The variables available are listed below. These variables can be added to your own analysis nTuples using the ANNPID TupleTool (same as ProbNN).

Variable	Comment
MC15TuneDNNV1	Deep Neutral Network
MC15TuneFLAT4dV1	Trained to retain efficiency flatness in 4D
MC15TuneCatBoostV1	CatBoost

Binning variables in Run 1 and Run 2

In Run 2, the PID calibration samples are selected at the trigger stage, within the TurCal Turbo stream. This is a significant difference relative to Run 1, where all PID calibration candidates were selected in the stripping.

One consequence of this is that there are now "online" and "offline" values for each variable in the Run 2 WGP nTuples. Variables calculated at the trigger stage are refrred to as **"online"**, and correspond to the values determined during the online reconstuction that runs within the PID trigger lines. Each variable is subsequently calcuated offline, and are referred to as **"Brunel"** variables within PIDCalib. For instance, the online momentum for a track is referred to as "P", while the offline equivalent is "Brunel_P".

An important exception is the **"nTracks"** variable, which is not correctly calculated in the Turbo stream. The ONLINE "nTracks" should **NOT** be used as a binning variable for Run 2 Turbo analyses. The correctly calculated SPD multiplicity **"nSPDHits"**, should be used. There are also known issues with online variables for the 2015-2016 electron calibration samples (due to the wrong treatment of bremsstrahlung), therefore only offline variables should be used for these samples. In case of a doubt, please contact the mailing list.

A summary of which binning variables you should use in PIDCalib is provided in the table below. The table covers Run 1 Stripping analyses, Run 2 Turbo analyses, and Run 2 Stripping analyses. Binning variables are specified on the command line in PIDCalib, as shown in the example in the section below. If you wish to run on fewer than 3 dimensions, you can specify empty strings for the y and z dimensions.

Analysis type	X variable	Y variable	Z variable
Run 1* (Stripping)	"P"	"ETA"	"nTracks"
Run 2 (Turbo)	"P"	"ETA"	"nSPDHits"
Run 2 (Stripping)	"Brunel_P"	"Brunel_ETA"	"nTracks_Brunel" or "nSPDHits"

* The distinction between online and offline does not apply in Run 1 data, hence why "Brunel" does not appear in the variable names.

Binning Scheme

The PIDCalib Binning Optimizer helps the user to choose a binning scheme that accurately captures changes in the PID efficiency for a given cut in bins of the PID binning variables.

An algorithm developed by the R(K*0) analysis was ported to python and PIDCalib to perform this task. It can be found under PIDCalib/PIDPerfScripts/scripts/python/BinningOptimizer. A detailed documentation and an example configuration file for this script can be found in the Urania PIDCalib GitLab repo.

It must be said that the configuration file gives example values which are reasonable for a certain decay and PID cut. In any case the user has to figure out an reasonable set of parameters to get a binning scheme, which is appropriate to a certain PID requirement and decay type.

Typical use example

Before you begin, it is useful to know the following. To save time, you can run different cuts on the same sample together. Instead of running the code several times for "[DLLp > 4.0]", "[DLLK > 4.0]" e.t.c., you can do "[DLLK > 10, DLLp > 5, MC15TuneV1 _ProbNNpi > 0.4]". This will produce Kaon ID, K->p mis-ID and K->pi mis-ID all in one go, if running on a kaon sample.

To run PIDCalib as normal, run the following command from within UraniaDev _v8r0:

```
python PIDCalib/PIDPerfScripts/scripts/python/MultiTrack/MakePerfHistsRunRange.py "Turbo16" "MagU
```

Remember to choose X, Y and Z binning variables as they apply to your analysis (see table above).

Within the Run 2 WGP nTuples, there are also additional new samples available. They can be used by specifying the following particle names on the command line:

Particle	PIDCalib name	Additional PIDCalib options
Kaon	K	K_DsPhi
Pion	Pi	Pi_KS
Proton	P	P_LcfB, P_IncLc
Muon	Mu	Mu_B_Jpsi, Mu_nopt (mu from prompt Jpsi with no pT cut)
Electron	e_B_Jpsi	

More information on /MakePerfHistsRunRange.py

This script makes an efficiency histogram, i.e. the PID efficiency for a given PID requirement, as a function of 1, 2, or 3 kinematic variables, for a specified sample version (data taking year), magnet polarity and particle type. A single .root file is produced containing a separate TH*F efficiency histogram for each PID requirement presented in the list of PID cuts and for each file in the calibration dataset. The efficiency histograms from each calibration file are averaged over to produce a histogram with the suffix _All in the output .root file.

By default, the script analyses data from all available runs. For testing it can be useful to only run over a few of the calibration files, this can be done by cutting on the "runNumber" variable using the -c option to apply a cut. Further options can be seen by calling the script with the '-h' option.

The default kinematic binning defined in this script involves a 3D binning in momentum (<latex>p</latex>), pseudorapidity (<latex>\eta</latex>) and track multiplicity (nTracks), but the analyst is free to choose a lower dimensional binning if they prefer (i.e. 2D or 1D) with any arrangement of bins they desire. Analysts should be aware that MC does not correctly reproduce the nTracks distribution of data. Hence it is not advisable to use a 3-D binning if the subsequent reference sample is going to be standard MC.

It is possible to bin in any variable which the dataset contains, as defined in TupleDataset.py above. An example of how to create the user defined binning schemes can be found in MultiTrack /ex_customBinning.py. Calibration samples should be generated specifying the python script containing these custom binning schemes and the binning scheme to MakePerfHistsRunRange.py as in the following example (note that you have to configure it properly adding the rest of the required flags and arguments):

```
python $PIDPERFSCRIPTSROOT/scripts/python/MultiTrack/MakePerfHistsRunRange.py --binSchemeFile="cu
```

After this step, the binning scheme and the track type you are calibrating also have to be passed to PerformMultiTrackCalib.py for the calibrations, as in the following example (as before, note that you have to configure the script properly with the rest of the required flags and arguments):

```
python $PIDPERFSCRIPTSROOT/scripts/python/MultiTrack/PerformMultiTrackCalib.py -s "trType" "custo
```

where "trType" has to be replaced by any of the valid track types: "K", "P", "Pi", "e_B_Jpsi" or "Mu".

The syntax for some example PID cuts are as given in the following table. Cuts involving functions of multiple variables are also perfectly acceptable, e.g.

```
'[DLLK > 5.0 && DLLe > 0.0, log(V2ProbNNK /V2ProbNNpi) > 2.0]'
```

PID Variable Name in WGP nTuple	Description	Example syntax
PIDK	Combined <latex>\Delta\log\mathcal{L}_{K-\pi}</latex>	DLLK > 2.0
PIDp	Combined <latex>\Delta\log\mathcal{L}_{p-\pi}</latex>	DLLp > 2.0
PIDmu	Combined <latex>\Delta\log\mathcal{L}_{\mu-\pi}</latex>	DLLmu > 2.0

PIDe	Combined <latex>\Delta\log\mathcal{L}_{e-\pi}</latex>	DLLe > 2.0
MC15TuneV1 _ProbNNk	NeuroBayes Bayesian posteriori probability: <latex>K</latex>	MC15TuneV1 _ProbNNK > 0.2
MC15TuneV1 _ProbNNpi	NeuroBayes Bayesian posteriori probability: <latex>\pi</latex>	MC15TuneV1 _ProbNNpi > 0.2
MC15TuneV1 _ProbNNp	NeuroBayes Bayesian posteriori probability: <latex>p</latex>	MC15TuneV1 _ProbNNp > 0.2
MC15TuneV1 _ProbNNe	NeuroBayes Bayesian posteriori probability: <latex>e</latex>	MC15TuneV1 _ProbNNe > 0.2
MC15TuneV1 _ProbNNmu	NeuroBayes Bayesian posteriori probability: <latex>\mu</latex>	MC15TuneV1 _ProbNNmu > 0.2
isMuon	Track passes IsMuon requirement	IsMuon ==1.0
isMuonLoose	Track passes IsMuonLoose requirement	IsMuonLoose ==1.0
InMuonAcc	Track is in Muon acceptance	InMuonAcc ==1.0
hasRich	Track has RICH hits	HasRich ==1.0
nShared	Number of tracks with shared hits in the Muon stations	nShared>3

The default binning has been tuned to the PID performance vs. <latex>p</latex>, <latex>\eta</latex> and nTracks of the <latex>\Delta\log\mathcal{L}_{K-\pi}</latex> variable for the full standard Kaon and Pion samples. This binning is unlikely to be optimal for other PID variables, so you will probably need to adjust the binning for your chosen variable. The binning may also suffer if you are only looking at a sub-sample of tracks in each sample e.g negative, or a sub-sample of the run range, or the muon unbiased samples which have lower statistics.

Depending on your analysis you may need to apply "hasRICH" requirements to the calibration samples prior to extracting the efficiency. This can be done via the -c option. Add

```
-c "HasRich = 1.0"
```

as an additional argument to the script. For Run 2, please use the offline variable (

```
"Brunel_HasRich = 1.0"
```

) if you work with the Stripping output.

Note the need to specify the particle type as the variable name e.g. (K_HasRich) is **no longer required**.

Tips on muon identification and mis-identification performance

Depending on your analysis you may need to apply "InMuonAcc" requirements to the calibration samples prior to extracting the efficiency; if you don't apply any requirements to the muon sample, you are measuring the combination of the Muon System acceptance and efficiency. To measure the efficiency only you have to use the -c option. Add -c "InMuonAcc == 1.0" as an additional argument to the script.

To measure the mis-identification rate of hadrons going to muon you have to apply a "trigger unbias condition". This ensures that the hadron samples being used to measure the mis-ID rate did not trigger the event, thus decoupling any trigger efficiency effect from the mis-ID rate of interest.

To measure K/pi to mu rate you have to use the -c option, adding the -c "MuonUnbiased = 1.0" additional argument to the script for Run 2 data. For Run 1 data, you should add -c "Unbias_HLT1 = 1.0". To measure the p to mu rate, you have to use the same syntax.

Electron mis-identification and trigger bias

When measuring the h -> e mis-ID rate, one must ensure that the hadron calibration samples did not trigger the event. To apply such a "trigger unbias condition", you can add -c "ElectronUnbiased = 1.0" when running PIDCalib to create your mis-ID rate histograms. This is similar to the "MuonUnbiased" condition detailed above for h -> μ mis-ID.

Measuring the efficiency with /PerformMultiTrackCalib.py

This script takes as input the performance histograms produced by MakePerfHistsRunRange.py, together with a reference sample (either data or MC) that possess the kinematics of the signal decay which the analysts is wanting to determine the PID efficiency for. This script then determines both the individual track, as well as the overall, efficiencies for each event in the reference sample (i.e. per event PID weights). The script then determines the PID efficiency averaged over all events in the reference sample, this is the efficiency that you will probably want to quote for the PID cut. The method by which this calibration is performed is that as described on pages 43 - 46 of the charm cross-section ANA.

In the reference nTuple you only need to have the variables in which you bin, but of course you can have more if you like. For the standard binning scheme this is momentum, pseudo-rapidity and the number of tracks. Putting these into your ntuple is described in this section.

In the latest release of the PIDCalib packages, the PerformMultiTrackCalib.py script takes a list of arguments like the MakePerfHistsRunRange.py script does. The latest version of this script can be run with the following commands:

```
python $PIDPERFSCRIPTSROOT/scripts/python/MultiTrack/PerformMultiTrackCalib.py
```

You should see the following output.

```
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
    __    __  ________        ____  ________  ______      ___ __
   / /   / / / / ____/ /_    / __ \/  _/ __ \/ ____/___ _/ (_) /_
  / /   / /_/ / /   / __ \  / /_/ // // / / / /   / __ `/ / / __ \
 / /___/ __  / /___/ /_/ / / ____// // /_/ / /___/ /_/ / / / /_/ /
/_____/_/ /_/\____/_.___/ /_/   /___/_____/\____/\__,_/_/_/_.___/

\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

error: too few arguments

usage: PerformMultiTrackCalib.py [-h] [-i DIR] [-X NAME] [-Y NAME] [-Z NAME]
                                 [-s TYPE NAME] [-q] [-N] [-x NAME] [-y NAME]
                                 [-z NAME] [-P NUM] [-w] [-W NAME]
                                 <stripVersion> <magPol> <refFileName>
                                 <refTree> <outputFilename> <track>
                                 [<track> ...]

Perform the PID calibration for a given:
        a) stripping version, <stripVersion> (e.g. "20")
        b) magnet polarity <magPol> ("MagUp" or "MagDown")
        c) reference (i.e. signal or control) sample <refTree>,
           located in file <refFilename>

The per-event PID efficiencies will be stored in the ROOT file <outputFilename>.
The remaining positional argument, <track>, specifies the reference track,
and takes the form [<trackName>, <trackType>, <pidCut>], where <trackName>
is the name of the reference track in the TTree, <trackType> is the type of
track and <pidCut> is the PID cut to evaluate for that track.

If the reference sample contains multiple final-state tracks, the remaining tracks
can be specified as additional arguments.

Valid track types are: "K", "P", "Pi", "e" or "Mu".

For a full list of arguments, do: 'python PerformMultiTrackCalib.py -h'

e.g. python PerformMultiTrackCalib.py "20" "MagUp" "$HOME/MyAnalysis/MySignalSample.root" "Signal
    "MyPIDResults.root" "[Kaon,K,DLLK>4.0]" "[Pion,Pi,DLLK<4.0]"
```

Inside the [Pion,Pi,DLLK<4.0] argument, "Pion" refers to the particle branch name in your signal sample .root file.

If certain variables do not have the same name in the PIDCalib calibration samples and your signal reference nTuples, you can specify the nTuple branch names explicitly by setting -x "my_branch_X" -y "my_branch_Y" -z "my_branch_Z". Here, "my_branch_X", "my_branch_Y" and "my_branch_Z" represent the names of your signal nTuple branch names. For example, for a reference track called "part" in your signal nTuple, with branch names "part_P" and "part_ETA", you should specify: -x "P" -y "ETA".

Note the use of lowercase letters -x, -y and -z here. These arguments should be specified in addition to the -X, -Y and -Z arguments, which specify the binning dimensions of the calibration samples.

Running PIDCalib scripts in batch jobs

PIDCalib requires a Grid proxy, in order to access the bookkeeping information which is used to find the correct path to the calibration files. In order to run batch jobs for PIDCalib, you need to generate a proxy interactively which your lxbatch jobs can just use, since they cannot generate one themselves. You should set the X509_USER_PROXY environment variable to save the proxy somewhere sharable, like your AFS home area:

export X509_USER_PROXY=/path/to/your/home/area/.grid.proxy

Then use the standard LHCb command to create a proxy. One tip is to add something like "-v 120:00" to generate it for longer than the default 24 hours. The above is for 120 hours (there is a max on how long it can be).

Then make sure the above variable is also defined in your batch jobs. One way is to just put the above in your shell login profile, which means it will be applied to all your subsequent logins.

A typical batch script might look like:

```
#!/bin/bash

source /cvmfs/lhcb.cern.ch/group_login.sh

export HOME=/home/$USER/

export CMTCONFIG=x86_64-slc6-gcc62-opt

export X509_USER_PROXY=/home/$USER/.grid.proxy
```

```
version=UraniaDev_v8r0

path=/path/to/your/PIDCalib/${version}/

${path}/run python ${path}/PIDCalib/PIDPerfScripts/scripts/python/MultiTrack/MakePerfHistsRunRang
```

which can be submitted to Condor using Ganga:

```
j = Job()

j.application.exe = File( batch.sh')

j.backend = Condor()

j.backend.getenv = "True"

j.backend.cdf_options['+JobFlavour']='"nextweek"'

j.submit()
```

Note that if you've installed PIDCalib using the setup options above, you should pay attention to the platform which you compiled for and adjust this script accordingly.

Where can I find the calibration samples?

The paths to the calibration samples both for Run 1 and Run 2 can be found in this link.

Where can I see the fit results for each calibration sample?

The fit results are available in a web page, where Run2 pp, pA and Ap folders are collected: https://lhcb-pid-wgp-plots.web.cern.ch/lhcb-pid-wgp-plots/Run2/.

PID Resampling

At present the only supported tool for PID resampling is Meerkat, details of which can be found here and in this presentation. This package has been migrated to Urania and housed under PIDCalib.

The support for MCResampling tool has now been stopped and the information archived.

Advice on systematic uncertainties

When evaluating systematic uncertainties on PIDCalib-derived quantities, several underlying assumptions of the approach should be considered.

1) Global assumptions

a) Efficiency parametrisation

The efficiencies measured by PIDCalib using calibration samples are provided in bins of track kinematics (P, ETA) and event multiplicity. The PID performance is known to depend strongly on these variables, but additional dependence on other quantities is also possible. For instance ProbNN, which is an MVA-based measure that uses several candidate and event-level variables as input, could depend more strongly on track quality for certain cuts (see here).

When considering h -> μ mis-ID, genuine decays in flight are also likely to have poorer track quality. Such cases may thus have a different PID performance compared to non-decaying hadrons. If your analysis is sensitive to h -> μ mis-ID and/or decays in flight, we recommend using the "MuonUnbiased" and "isMuon" flags in the calibration data and your own signal samples to investigate such instances.

Studies of the hadron calibration data subsets passing isMuon and MuonUnbiased will be carried out.

b) Track independence

PIDCalib assumes that the total PID efficiency for a signal decay is given by the product of the individual track PID efficiencies i.e. epsilon_tot = epsilon_1 * epsilon_2 * ...epsilon_n, where 1,2...n represent the n signal tracks.

For a given decay, the signal track kinematics will be correlated to some degree. The RICH rings for each track could thus overlap, resulting in an inter-dependence of the track PID performance. Depending on the degree of kinematic correlations, this may violate the track independence assumption.

To assign a systematic uncertainty for this effect, signal MC can be used. The true efficiency given by epsilon_tot = N(pass all cuts) / N(tot) can be compared with epsilon'_tot = (N(pass cut 1) * N(pass cut 2) * ... N(pass cut n)) / N(tot)^n to determine the effect of efficiency factorisation. Although the absolute efficiencies calculated on MC are incorrect, the difference in efficiency between these methods can still be used as a measure of the systematic.

c) Overlap between calibration and signal track kinematics

Efficiencies and mis-ID rates can only be determined for tracks which fall inside the kinematic coverage provided by the calibration samples. Thus, users should ensure that their signal track passes the same track P, PT, and ETA requirements as any of the calibration samples they use for this track. This applies both for efficiency and mis-ID calibration samples used.

For example, you may wish to use the muon calibration sample to calibrate muon -> hadron mis-ID. The muon calibration sample has a minimum PT cut of 800 MeV, and so your signal hadron tracks should also have this requirement applied. There is a dedicated muon sample without this minimum PT cut to provide further coverage.

If your signal tracks fall outside the PIDCalib sample coverage, you should ask:

- What percentage of my tracks fall outside the coverage range? What would the impact on statistical precision be if I removed those tracks?

- If I keep the tracks and apply a PID cut to them, what will the systematic uncertainty on the efficiency be? Note that this systematic should be conservative, since there is no calibration information available.

Ultimately users should determine whether the loss of precision due to removing tracks, or the additional systematic from non-calibrated efficiencies, is dominant for their analysis.

2) Method assumptions

a) Efficiency within each bin is constant

Track kinematics are not constant within a given bin of kinematics. In addition, the kinematic distributions for your signal tracks and the calibration sample tracks will not be identical within a bin. This can lead to differences in efficiency between the calibration and signal samples for a given bin.

This effect can be mitigated by choosing a sensible binning that minimises the differences between your signal and the calibration sample kinematics within each bin. Studying the variation across several binning schemes can then be used to determine an appropriate systematic uncertainty.

b) sWeight method - global fit to calibration samples over all kinematics

Dedicated studies of the global sWeight method have indicated per-track systematic uncertainties at the 0.2% level. However, the uncertainty could be larger for specific PID cuts and in specific regions of phase space.

These effects are being investigated, and some obvious improvements can be implemented, for example performing fits in bins of kinematics.

3) Summary

Please take each of the above factors into consideration, and ensure that the PIDCalib systematics assigned in your analysis are not unreasonably small. If you have any doubts about the uncertainties you have assigned, please ask for advice on the mailing list (lhcb-phys-pid-calibration@cernNOSPAMPLEASE.ch).

Run 1 archived information

V4 tuning available for RUN I data

https://indico.cern.ch/event/537782/contributions/2194458/attachments/1286984/1915101/PIDCalib.pdf

The electron problem in the slides has been solved.

An example of how the run the code is:

```
python PlotCalibDistributions.py "21_MCTuneV4" "MagUp" "K" V4ProbNNpi
```

Sneha's suggestion on merging proton sample in Run1

You divide your phase space into three regions:

(1) Where only the L0 sample is used

(2) Where only the Lc sample is used

(3) Where both are combined.

You can then find appropriate binnings for the 3 regions, and implement them in PIDCalib. In general you will be able to use something much finer for region (1) . You can then merge the output for (2) from the two samples. For (3), a statistical combination can be made.

Some more information is available in her talk:
https://indico.cern.ch/event/331664/session/0/contribution/6/attachments/646367/889108/PID_protons_forANAweek.p

Changes to protons in Run 2

In Run2 dedicated triggers and prescales were implemented to achieve a significantly better coverage in L0 sample than was present in Run 1. As a result, it is expected that the coverage in the L0 sample is sufficient for most analyses, and furthermore now has high yields and high purities meaning that it is a good and reliable calibration sample. The Lc samples in Run2 have not yet been validated. If they are used in conjunction with L0 samples a report should be given at the PID meeting comparing performance and detailing the specific gains achieved by using both.

Stripping 21 electrons

Issues with these samples are now resolved and the new corrected samples are available. As a reminder, the performance is dependent on whether or not the electron candidate has brem photons added or not. It is possible to split the calibration sample via the cut command -c "HasBremAdded==1" etc. There is also a dependence with the calo region, which is also correlated to the change in performance with pseudorapidity. Depending on your analysis needs, it may be better to bin in calo region rather than pseudorapidity. This has not been studied in depth, so if you do look into this please report your findings at a PID meeting!

v2 vs v3

To get different versions of ProbNN in your tuple, try "TupleToolANNPID". (Default one in other code, like LoKi, is V2)

V3 is not an exact 'upgrade' of V2, but it adds more kinematic regions and removed ghosts from the training samples. For many decays it will be better, but there might be some specific cases where it is not. So it is worth trying both and comparing their performance!

More information can be found at:

https://twiki.cern.ch/twiki/bin/view/LHCb/GlobalParticleID#Combined_variables_DLL_and_ProbN

5TeV samples now available

The 5 TeV samples are now available for all hadrons and muons. Use "5TeV" in the command line to access the samples. PIDPerfScripts v10r1 or higher is required.

Migration to Urania v4r0

Migration to Urania v4r0 was not seamless. Please report any strange behaviour to the mailing list. The PIDtables functionality is temporarily disabled since this is not yet compatible. If anyone actually misses these tables, we would be interested to hear from you.

Stripping 23 samples now available

The Stripping 23 samples are now available for all hadrons and muons. This is data collected in 2015 since the September technical stop. Use "23" in the command line to access the samples. PIDPerfScripts v10r0 or higher required. (v9r6 was missing an essential directory). This is the sample to use for S23r1, since there was no change in reconstruction of any subdetector.

Stripping 21 samples now available.

The Stripping 21 and 21r1 samples are now available for all species. Use "21" or "21r1" in the command line to access the samples. PIDPerfScripts v9r5p1 or higher required.

Proton Samples available for Run I

Look at this page for details: Protons for Run1

PIDCalib use for early measurements.

Look at this page: Test instructions

Deprecation of the variable - June 2015

Versions of PIDPerfTools v7r0 and higher no longer contain the DLLpK variable. The reasons are due to incompatibility with simultaneously retaining this variable and having access to both online and offline variables. Since this variable was no more than a difference in two other variables analysts can still use it by calling "[DLLp-DLLK >xx]". This is not dissimilar to how a number of PIDcuts are now some choice of a mathematic function of a number of variables.

Problems in.py - May 2015

The script is found to be problematic and has been removed from versions of PIDPerfScripts 9 and above. For information on calculating the statistical uncertainties (which should rarely be the largest source of uncertainty) see the section on uncertainties.

Mock up of PID samples for EM - Feb 2015

Using the number of calibration events collected for each PID sample in Run1, define a Run2-equivalent sample. Specifically: from 2012 data, count the number of tracks on each sample, divide by 2000 (2 fb-1), obtain the number of calibration tracks per pb-1 after ALL selection, and finally, multiply by 2 for roughly taking into account the effect of 8 TeV --> 13 TeV energy increase.

In the table below is indicated, for each particle and for each magnet polarity, the run interval that corresponds of the number of calibration tracks expected in 1 pb-1 of Run2. (Use the min/max Run syntax to produce the efficiency table from the selected interval: python MakePerfHistsRunRange.py   minRun=117846   maxRun=117848)

Particle	Down		Up	
	Min	Max	Min	Max
Muon	117846	117848	125962	125965
Kaon/Pion	126915	126933	125951 (just one run)	
Proton	117849	117851	120241	120317

New Samples containing v3 variable - Nov 2014

A new set of calibration samples is available. New variables are stored including the v3 ProbNN variables. The twiki page is updating information to v8 of PIDPerfScripts. Some of the available features may not be present in v7 and below.

New Proton Samples - August 2014

A new calibration sample of protons is available. These are protons from Lc decays and although the statistics are low in comparison to those coming from Lambda the coverage in momentum and eta is different. To access these sample the particle name is "P_IncLc". These samples should be treated as in "beta-release" while an increase in statistics via an additional stripping line is explored. Please report any strange behaviour relating to these samples.

Better handling of negative efficiencies - August 2014

Change in calculation of efficiencies:

- Efficiency calculated by dividing the sum of the total no of tracks passing cut by total no of tracks. These totals relate to the sum over the desired run range
- Previous calculation computed the efficiency for each sub-sample of data and then averaged over the sub-samples to give the total efficiency
- Advantage of new method is that bins where efficiency are close to 0 have a better treatment of uncertainties
- Efficiency in any given bin may now be slightly different than previously. However this difference should be less than the statistical uncertainty from the calibration sample unless the bin in question has a very low efficiency.

Faster PIDCalib with electrons - May 2014

A new release has been made of the PID Calib software which runs about 2.5 times faster and has several new features. The new software will not change the result of any previous PID Calib study, so it not necessary to re-do any studies with the latest version. It does however add an electron sample which can be used to measure electron PID efficiencies in data.

Changes from earlier datasets

These datasets include an electron sample and restore previously missing data and use MCTuneV2 for the ProbNN variables by default. It is no longer necessary to add "MC_TuneV2" to any script argument. In addition, RICH threshold variables are available in these samples:

- Kaon and Pions from Dstars
- Protons from Lambda0 decays
- Protons from Lc decays
- Muons from J/Psi decays

The calibration tracks have not had requirements on "hasRich", and the muon tracks do not have requriments to be "InMuonAcc" as part of the preselection. If your signal decay does have these requirements you should apply a preselection cut to the calibration samples. This can be done using the -c option of MakePerfHistsRunRange.py. The particle specifier is no longer required to start the cut variable. ie. use -c "InMuonAcc==1"

It is no longer needed to have a copy of the PIDCalib/CalibDataScripts package. This has been moved to expert-only.

The dedicated muon unbias samples have not been updated. Instead additional flags are added to the hadron samples which can be cut on to select tracks that passed certain trigger requirements.

Running the Code

The PIDPerfScripts package provides a series of generic python scripts to satisfy the calibration demands of a typical analyst. To confirm your environment is correctly configured for running the code, attempt the following:

```
python $PIDPERFSCRIPTSROOT/scripts/python/MultiTrack/MakePerfHistsRunRange.py
```

If correctly configured, you should get the following output: (note here that the arguments are missing)

```
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
    __    __  __________        ____  ________  ______      ___ __
   / /   / / / / ____/ /_      / __ \/  _/ __ \/ ____/___ _/ (_) /_
  / /   / /_/ / /   / __ \    / /_/ // // / / / /   / __ `/ / / __ \
 / /___/ __  / /___/ /_/ /   / ____// // /_/ / /___/ /_/ / / / /_/ /
/_____/_/ /_/\____/_.___/   /_/   /___/_____/\____/\__,_/_/_/_.___/

\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

error: too few arguments

usage: MakePerfHistsRunRange.py [-h] [-x NUM] [-y NUM] [-f NUM] [-c CUTS]
                                [-o DIR] [-X NAME] [-Y NAME] [-Z NAME]
                                [-s NAME] [-b NAME] [-q] [-T] [-l] [-t FILE]
                                [-M]
                                <stripVersion> <magPol> <partName> <pidCut>

Make performance histograms for a given:
        a) stripping version <stripVersion> (e.g. "20")
        b) magnet polarity  <magPol> ("MagUp" or "MagDown")
        c) particle type <partName> ("K", "P", "Pi", "e" or "Mu")
        d) PID cut, <pidCut>
Multiple PID cuts can be specified if necessary, e.g. "[DLLK > 0.0, DLLK > 4.0]".
```

```
In this case, a performance histogram will be produced for each PID cut.

For a full list of arguments, do: 'python MakePerfHistsRunRange.py -h'

e.g. python MakePerfHistsRunRange.py  --minRun=114205 --maxRun=114287 "20" "MagUp" "K" \
     "[DLLK > 0.0, DLLK > 4.0 ]"
```

⚠ For all samples (as of Dec 2013) you need only specify the stripping version as "20" or "20r1". These contain both V2 and V3 tunings of ProbNN

Uncertainties

Statistical uncertainties - there are two sources of statistical uncertainty that coming from the calibration sample size and that coming from the reference sample size. It is expected that the statistical uncertainties will (unless the reference sample is small) be smaller than the systematic uncertainty. Note also that the binning systematic is likely to increase if the calibration sample size decreases since you have to move to wider binning.

There is currently no script to calculate the statistical uncertainty from the calibration sample. The script is planned. (Feel free to help) If you wish to do it the process is simple. Generate n different efficiency histograms where the efficiency in each bin is a random value drawn from a gaussian with mu and sigma according to the bin content and bin error from the default histogram produced by MakePerfHistsRunRange.py. Calculate the average event efficiency for your reference sample according to each of the smeared histograms. The width of the average efficiency is the statistical uncertainty due to the caibration sample. When smearing you should take into account if you have any bins near or past the physical limit (i.e efficiency greater than 1, less than 0) and take appropriate action.

For K/pi at least a 0.1% systematic uncertainty should be assigned to the sWeighting method

Another thing to consider might be, does changing the binning scheme significantly change the efficiency?

Yet another important consideration, perhaps most of all, is the match between the reference tracks and the tracks of your signal decay in data. In particular the MC does not correctly reproduce the nTracks distribution. In most cases the recommended course of action if using MC reference samples is to employ a 2-D binning scheme which essentially integrates over the nTracks distribution. Or you could reweight the distribution, but consider the additional uncertainties associated with reweighting.

Separating Track Charges

It is possible to select a certain track charge. If using the standard python scripts the selection is passed via a command line argument. To select only positively charged tracks add -c "trackcharge = 1"=to the list of arguments. To select only negatively charged tracks add -c "trackcharge = -1"=to the list of arguments. The variable listed as "Charge" can also be used but it has 0 for positive and 1 for negative and hence is less intuitive.

Which calibration sample to use?

For current or upcoming analyses the argument to provide as the stripping version is either "20", "20r1", "21", "21r1" or "22" or "23". The best Run 1 Lc samples for protons are only available on "21" or "21r1". "22" is the correct version to use for forthcoming early measurements. "23" is for 2015 data after the Sept TS. This covers Stripping 23r1 and most likely Stripping 24 (tbc). If you need to access older samples, e.g those of S17 or earlier, or those containing MCTuneV1 please see below. These samples are outdated so only use with good reason. It is possible that not all the samples survived the migration to EOS. If you think this is the case please email smalde@cernNOSPAMPLEASE.ch.

samples - if you still use these please email smalde@cernNOSPAMPLEASE.ch

The tagged versions CalibDataScripts (v4r3) and PIDPerScripts (v6r1) should be used.

The MCTuneV1 S20r1 samples are available at /castor/cern.ch/grid/lhcb/user/p/phunt/CalibData and /eos/cern.ch/grid/lhcb/user/p/phunt/CalibData. To use them set the following variables. More details on how to do this can be found in this section

```
export CALIBDATASTORE=eoslhcb.cern.ch//eos/cern.ch/grid/lhcb/user/p/phunt/CalibData

export MUONCALIBDATASTORE=eoslhcb.cern.ch//eos/cern.ch/grid/lhcb/user/p/phunt/CalibData

export MUONCALIBREFDATASTORE=eoslhcb.cern.ch//eos/cern.ch/grid/lhcb/user/p/phunt/CalibData
```

Earlier MCTuneV1 RICH calibration samples (S17 and S20) are stored at /castor/cern.ch/grid/lhcb/user/p/powell/CalibData and /eos/cern.ch/grid/lhcb/user/p/powell/CalibData, while the muon calibration samples are stored at /castor/cern.ch/grid/lhcb/user/j/jotalora/CalibData and /eos/cern.ch/grid/lhcb/user/j/jotalora/CalibData To use them set the following variables. More details on how to do this can be found in this section

```
export CALIBDATASTORE=eoslhcb.cern.ch//eos/cern.ch/grid/lhcb/user/j/powell/CalibData

export MUONCALIBDATASTORE=eoslhcb.cern.ch//eos/cern.ch/grid/lhcb/user/j/jotalora/CalibData

export MUONCALIBREFDATASTORE=eoslhcb.cern.ch//eos/cern.ch/grid/lhcb/user/j/jotalora/CalibData
```

Archived Calibration Samples

All previous calibration samples have now been archived and are stored on the following CASTOR locations:

```
/castor/cern.ch/user/p/powell/CalibData/Reco08
                                       /Reco09
                                       /Reco10
                                       /Reco11
```

Old samples Summary Table

Samples are stored in various locations. To point to the correct location you can change the $CALIBDATASTORE or $MUONCALIBDATASTORE. Details here: this section.

The table below summarizes available datasets and gives some comments to their usage.

Data Sample	Code Versions	Castor user area	Script Call	Comments
S20 or S20r1 MCTuneV2: K, Pi, P	CalibDataScripts v4r4 , PIDPerfScripts v6r2p2 or v6r2p1	...castor.../smalde/....	Call regular scripts with "20" or "20r1	***Standard Sample***
S20 or S20r1 MCTuneV2:mu,	CalibDataScripts v4r4, PIDPerfScripts v6r2p2	...castor.../smalde/...	Call regular scripts with	***Standard Sample***

muon unbiased			"20" or "20r1"	
S20r1 MCTuneV1 all types	CalibDataScripts v4r3, PIDPerfScripts v6r1	...castor.../phunt/...	Call regular scripts with "20" or "20r1"	**Only** use if you really want MCTuneV1. Full dataset unavailable
S20 MCTuneV1 K,Pi,P	CalibDataScripts v4r3, PIDPerfScripts v6r1	...castor.../powell/...	Call regular scripts with "20" or "20r1"	**Only** use if you really want MCTuneV1.
S20 MCTuneV1 Muon	CalibDataScripts v4r3, PIDPerfScripts v6r1	...castor.../jotalora/...	Call regular scripts with "20" or "20r1"	**Only** use if you really want MCTuneV1.
S17 Muon	CalibDataScripts v4r3, PIDPerfScripts v6r1	...castor.../jotalora/...	Call regular scripts with "17"	
S17 or earlier K, Pi, P	CalibDataScripts v4r3, PIDPerfScripts v6r1	...castor.../powell/...	Call regular scipts with "17" etc	May be code incompatibilites with samples earlier than S17

Location of the calibration samples

The calibration samples for kaons, pions, protons and muons are available on EOS. If you wish to copy these files over to your local institute, you are free to do so. The following environmental variables are defined in $PIDPERFSCRIPTSROOT/cmt/requirements:

- CALIBDATASTORE: specifies the top-level directory of all calibration samples;

By editing these variables, the user can inform the scripts where to look for the calibration samples.

After edit the requirements file, the user must set up the environment again to pick up the changes, i.e. the user should do the following:

```
cd $PIDPERFSCRIPTSROOT/cmt

emacs -nw requirements

SetupUrania
```

Where 'emacs -nw' should be replaced by the user's preferred text editor.

Using the head of PIDCalib

If you need to use the head of the PIDCalib package, either because the PIDCalib maintainer asks you to, or because you wish to commit changes to the package, you can do the following:

```
SetupUrania --build-env getpack PIDCalib/PIDPerfScripts head getpack PIDCalib/PIDPerfTools head c
```

⚠ Always contact the maintainer of the package before committing any changes.

Variables stored in the calibration datasets

These are the varibles which can: be cut on before the dataset is made, have their efficiency measured by MakePerfHistsRunRange.py or be a variable which is binned in.

Var name	Description
Charge	+ve/-ve charged track (see this section)
trackcharge	a more user friendly version of Charge
nTracks	Total number of tracks in Best container
runNumber	runNumber of event
DLLK	Combined <latex>\Delta\log\mathcal{L}_{K-\pi}</latex>
DLLp	Combined <latex>\Delta\log\mathcal{L}_{p-\pi}</latex>
DLLe	Combined <latex>\Delta\log\mathcal{L}_{e-\pi}</latex>
DLLmu	Combined <latex>\Delta\log\mathcal{L}_{\mu-\pi}</latex>
V2(3)ProbNNK	Bayesian posteriori probability with either MCtuning v2 or v3: <latex>K</latex>
V2(3)ProbNNpi	Bayesian posteriori probabilitywith either MCtuning v2 or v3: <latex>\pi</latex>
V2(3)ProbNNp	Bayesian posteriori probability with either MCtuning v2 or v3: <latex>p</latex>
V2(3)ProbNNmu	Bayesian posteriori probability with either MCtuning v2 or v3: <latex>\mu</latex>
V2(3)ProbNNe	Bayesian posteriori probability with either MCtuning v2 or v3: <latex>e</latex>
V2(3)ProbNNGhost	Bayesian posteriori probability with either MCtuning v2 or v3: Ghost
P	Track momentum (<latex>p</latex>) [MeV/c]
PT	Track transverse momentum (<latex>p_{\rm T}</latex>) [MeV/c]
ETA	Track pseudo-rapidity (<latex>\eta</latex>)
PHI	Track phi angle (<latex>\phi</latex>) [rad]
IsMuon	Track passes IsMuon requirement
InMuonAcc	Track passes InMuonAcc requirement
IsMuonLoose	Track passes IsMuonLoose requirement
nShared	Number of tracks with shared hits in the Muon stations
RICHThreshold_pi	Track had enough momentum to pass the RICH <latex>\pi</latex> thresholed
RICHThreshold_p	Track had enough momentum to pass the RICH <latex>p</latex> thresholed
RICHThreshold_e	Track had enough momentum to pass the RICH <latex>e</latex> thresholed
RICHThreshold_K	Track had enough momentum to pass the RICH <latex>K</latex> thresholed
RICHAerogelUsed	Track passed RICHAerogelUsed requirements
RICH1GasUsed	Track passed RICH1GasUsed requirements
RICH2GasUsed	Track passed RICH2GasUsed requirements
HasRich	Track passes HasRich requirement
HasCalo	Track passes HasCalo requirement
CaloRegion	Region where track hit the CALO (0=Unkown, 1=BeamPipeHole, 2=Inner, 3=Middle, 4=Outer)
HasBremAdded	Track momentum has had bremsstrahlung photons added to it
nSPDHits	Number of hits in the SPD detector
TagTOS	Tag muon track passes certain trigger requirements
Unbias_HLT1	Kaon or Pion track passes certain trigger requirements at L0 && HLT1
Unbias_HLT12	K/Pi/P track passes certain trigger requirements at L0 && HLT1&& HLT2

Regarding TagTOS - this is when the tag particle passes is TOS with respect to L0Muon, and TOS on one of Hlt1(TrackAllL0 |TrackMuon|SingleMuonHightPT) and also TOS on Hlt2(SingleMuon |SingleMuonHighPT). Note specifically that this is the **tag** track of the Jpsi, while all other information on a track pertains to the probe. Use of this variable can be used as as systematic check on the impact of the muon trigger within the PID algorithms.

Recommened Variables for Reference Sample (i.e. to be contained in signal nTuple)

Var Name	Description
nTracks	Total number of tracks in Best container
nPVs	Total number of reconstructed primary vertices
particle_P	Track momentum (<latex>p</latex>) [MeV/c]
particle_PT	Track transverse momentum (<latex>p_{\rm T}</latex>) [MeV/c]
particle_ETA	Track pseudo-rapidity (<latex>\eta</latex>)
particle_PIDK	Combined <latex>\Delta\log\mathcal{L}_{K-\pi}</latex>
particle_PIDp	Combined <latex>\Delta\log\mathcal{L}_{p-\pi}</latex>

The variable 'particle_*' variables are accessible with the following LoKi::Hybrid::TupleTool code:

```
LoKi _All.Variables = { "P" : "P" ,"PT" : "PT" ,"ETA" : "ETA" ,"PIDK" : "PIDK" ,"PIDp" : "PIDp" }
```

The event variables 'nTracks' and 'nPVs' can be extracted using LoKi::Hybrid::EventTupleTool from DSTs like so :

```
myEventTupleTool.VOID_Variables = { "nTracks" : "TrSOURCE('Rec/Track/Best') &gt; TrSIZE " ,"nPVs"
```

or using the following lines for uDSTs:

```
myEventTupleTool.VOID_Variables = { "nTracks" : "RECSUMMARY( LHCb.RecSummary.nTracks , -1 )" ,"nP
```

or by adding the necessary TupleTool:

```
yourdecaytreetuple.addTupleTool("TupleToolRecoStats")
```

The exact name passed to the scripts can be specified by -xRefVarName etc

Utility scripts

Several other utility scripts also provided which analysts may find useful during the course of their investigations.

PID Performance Plots

PID performance plots, for the various reconstruction versions and datasets, can be found on the following RICH-Online twiki page. They are produced using the following script: $PIDPERFSCRIPTSROOT/scripts/python/PubPlots/MakeIDvsMisIDCurveRunRange.py

Converting the calibration dataset to to a TTree

The script $PIDPERFSCRIPTSROOT/scripts/python/Plots/CreateTTreeFromDataset.py converts a PIDCalib dataset into a TTree. This can be useful when debugging issues or just for plotting variables easily. It takes two arguments, the path of a PID Calib dataset and a path to save the new TTree to. The path of an input PID Calib dataset can be taken from the output of MakePerfHistsRunRange.py or you can search around in $CALIBDATASTORE and $MUONCALIBDATASTORE.

The latest version of this script can be run with the following commands:

```
python $PIDPERFSCRIPTSROOT/scripts/python/Plots/CreateTTreeFromDataset.py
```

```
usage: CreateTTreeFromDataset.py [-h] [-q] <inFile> <outFile>

Create a TTree from a PID Calib dataset. This is particularly useful when debugging.

For a full list of arguments, do: 'python CreateTTreeFromDataset.py -h'
e.g. python CreateTTreeFromDataset.py  inputFile.root outputFile.root
```

Plotting cut efficiency from the Calibration dataset as a function of a variable

It might be interesting to see if your PID cut efficiency is correlated with another variable in the PID Calib datasets. This script can be used to crate a histogram of the efficiency versus any variable in the dataset. An example might be in the case of electrons, where the efficiency of a PID cut could change as a function of the region of the calorimeter. To check this the command python PlotCalibEfficiency.py 20 MagDown e "[DLLe>2]" CaloRegion could be run to produce this plot.

The latest version of this script can be run with the following commands:

```
python $PIDPERFSCRIPTSROOT/scripts/python/Plots/PlotCalibEfficiency.py
```

```
usage: PlotCalibEfficiency.py [-h] [-x NUM] [-y NUM] [-f NUM] [-c CUTS]
                              [-n NAME] [-o DIR] [-s NAME] [-b NAME] [-q] [-M]
                              <stripVersion> <magPol> <partName> <pidCut>
                              <varName>

Make histograms of PID performance vs. momentum (or another varaible in the dataset) for a given:
        a) Stripping version <stripVersion> (e.g. "20")
        b) magnet polarity  <magPol> ("MagUp" or "MagDown")
        c) particle type <partName> ("K", "P", "Pi", "e" or "Mu")
        d) DLL or ProbNN cut <pidCut>
Multiple PID cuts can be specified if necessary, e.g. "[DLLK > 0.0, DLLK > 4.0]".
In this case, a performance plot will be produced for each PID cut.
        e) The variable of interest e.g "P" or "ETA"

For a full list of arguments, do: 'python PlotCalibEfficiency.py -h'
e.g. python PlotCalibEfficiency.py  --minRun=114205 --maxRun=114287 "20" "MagUp" "K" \
    "[DLLK > 0.0, DLLK > 4.0 && DLLe >0.0]" P
```

Plotting variables from the Calibration dataset

It might be interesting to plot variables inside the calibration dataset, for example the kinematics of the proton are known to not to match the kinematics of many signal decays very well. Using this script you can plot the momentum distribution (or any other variable) of the calibration sample. In the example of the protons above you could then compare the output of this script with your signal momentum distribution, to check the coverage of the calibration dataset.

The latest version of this script can be run with the following commands:

```
python $PIDPERFSCRIPTSROOT/scripts/python/Plots/PlotCalibDistributions.py
```

```
usage: PlotCalibDistributions.py [-h] [-x NUM] [-y NUM] [-f NUM] [-c CUTS]
                                 [-o DIR] [-s NAME] [-b NAME] [-q] [-M]
                                 <stripVersion> <magPol> <partName> <varName>

Plot the distribution of a variable from the PID Calib datasets, e.g. momentum, for a given:
        a) Stripping version <stripVersion> (e.g. "20")
        b) magnet polarity  <magPol> ("MagUp" or "MagDown")
        c) particle type <partName> ("K", "P", "Pi", "e" or "Mu")
        d) Variable to plot <variable>

For a full list of arguments, do: 'python PlotCalibDistributions.py -h'
e.g. python PlotCalibDistributions.py  --minRun=114205 --maxRun=114287 "20" "MagUp" "K" \
    "Pi" "[DLLK > 0.0, DLLK > 4.0 && DLLe > 0]"
```

Plotting variables from the Reference dataset

This script lets you plot variables in an ntuple using the same binning scheme as defined in the PID Calib package. This might be useful to do when deciding on a binning scheme to use. You could plot your reference dataset with the calibration dataset using the exact the same binning as you will be using to determine the PID efficiency.

The latest version of this script can be run with the following commands:

```
python $PIDPERFSCRIPTSROOT/scripts/python/Plots/PlotRefDistributions.py

usage: PlotRefDistributions.py [-h] [-n NAME] [-c CUTS] [-o DIR] [-X NAME]
                               [-s NAME] [-b NAME] [-q]
                               <partName> <varName> <refFileName> <refTree>
                               <refBranch>

Plot the distribution of a variable from a given reference sample using the PID Calib binning sch
        a) Particle type <partName> ("K", "P", "Pi", "e" or "Mu")
        b) Variable e.g P
a & b are used to define the binning scheme used
        c) <your file name> <your ref tree> <your branch name>


For a full list of arguments, do: 'python {0} -h'
e.g. python PlotRefDistributions.py "P_IncLc" "P" "PID_Modes.root" "IncLc2PKPiTuple/CalibPID" "La
```

Creating baseline tuples for the PID modes.

Occasionally there are requests for the baseline tuples that are used to make the PID datasets to be made available. In general these tuples store minimal information, and are large. Uploading these to grid/eos storage has not been forseen and is not part of the current workflow. However if you want to create your own tuples this is in practice quite easy following the instructions below. You will need to change the DaVinci version, platform, and make sure the bkk query is as you desire within the scripts. It should then be easy to see how to add the variables that you are interested in, and also to comment out the decay modes that you are not interested in.

SetupDaVinci build-env Getpack PIDCalib/CalibDataSel Cd CalibDataSet /cmt Cmt make Cd ../dev

The DaVinci script to make the ntuples is makePIDCalibNtuples.py and there is a ganga job script with some predefined optionsmakePIDCalibNtuples.ganga.py

In principle all you have to do is Ganga >*execfile(makePIDCalibNtuples.ganga.py)* >*PIDCalib.down12.submit()*

MCResampling tool (not mainteined)

The PID resampling can be found under the path:

```
PIDCalib/PIDPerfScripts/scripts/python/MCResampling
```

There are two scripts in this folder, namely MakePIDdistributionsRunRange.py and MakePIDCorrection.py. Instructions for how to use them can be found in this presentation, and also below.

.py

This script is similar to MakePerfHistsRunRange.py, but instead of returning PID efficiencies, it returns a file (for example PIDHists_Turbo15_MagUp.root) containing the PID distributions. This file is then used in MakePIDCorrection.py to perform the corrections.

To run this script, you can use the following command:

```
python MakePIDdistributionsRunRange.py "20" "MagUp" "[K,P]" "[DLLp,DLLK,ProbNNmu]" --cuts="[lab1:
```

where the final argument allows you to specify different sets of cuts, lab1 and lab2 in this example. Distribution histgrams for both sets of cuts will be stored in the ouptut file, under their own sub-directories.

MakePIDCorrection.py

This script corrects an input MC nTuple, using the output file from the above script. The corrected PID values for each event are saved in a new TTree under a new branch called e.g. K_PIDK_corr. The new TTree also contains all of the branches from the input TTree.

This script can be run using the command:

```
python -i MakePIDCorrection.py "MCfileToCorrect.root" "tuple/DecayTree" "[K:lab1/K,P:lab2/P]" "[P
```

where the option [K:lab1/K,P:lab2/P] specifies which set of cut conditions to apply to each particle (as specified when running MakePIDdistributionsRunRange.py).

-- Main.Sneha Malde - 10 Nov 2014

- PIDCalibTutorial.pdf: PID @ LHCb and a PIDCalib Tutorial

This topic: LHCb > PIDCalibPackage
Topic revision: r228 - 2020-11-04 - DanielCervenkov