Subject: Re: PID package
Posted by Bertram Kopf on Thu, 06 Aug 2009 12:44:30 GMT
View Forum Message <> Reply to Message

Dear Stefano,
I took a look into the new version of the PID packages and tried to figure out what is going on there.
I do not understand the code in all details and I have, therefore, a couple of questions and comments on it.
First of all, as you wrote in the posting above, "this will replace the old PndLhePidMaker and the PndMicroWriter". As far as I understand you and also that what I see in the code, the new PndPidCorrelator is responsible for doing some PID (whatever this means) and for creating the complete list of RhoCandidates
(or PndPidCandidates) which is the input for the analysis part. Am I right?
That means that this class should represent
a. the interface to the analysis (BTW: in my point of view such an interface is one of the most important part of the software) and
b. some PID related things/tasks

By looking a bit closer into the code I realized that the PndPidCorrelator does even more:
Apart from being the interface to the analysis part and for doing some PID related things also

c. an extrapolation of tracks to indiviual subdetectors (ToF, EMC, Mdt, DIRC, etc.) will be done and
d. detector specific PID related properties will be calculated there (e.g. truncated mean method for STT).
I think, this goes definitely far beyond that what should be done and should be provided in the PID related code. As you know the software has to be highly modular and flexible and has to make use of encapsulation in order to keep the code maintainable. This means that one has to decouple the different things which are currently done in the PndPidCorrelator and furthermore "non PID related" things should not be placed in the PID package at all.

In addition I have also some questions/remarks to technical points of the new code:

1. PndPidCorrelator:
 i. the implementation of the "singleton" has not been done properly. In case that you call the static method "PndPidCorrelator::Instance()" you will be get back a 0 pionter which could cause a crash in your application. In addition the constructor is defined there as "public" with the consequence that one can create such objects several times. There are a lot of documents available (on the web or books about design patterns) where one can find nice descriptions how to implement a singleton in a proper way.
 ii. in e.g. lines 758, 763,764 a division by zero has not been caught.
 iii. in line 436 a stack overflow has not been caught.

2. PndPidCandidate:
 What I see there is that lots of specific properties (in general doubles or integers) are copied to this object via "set methods". Why should this class not have just references to the relevant reco objects? Besides a better performance (avoidance of additional cpu time for several hard copies and of increasing memory) this would keep the code more flexible and maintainable.
 E.g. If in the software as it is right now right some relevant methods in one reco class will be removed or changed, you have also to modify the PndPidCorrelator and the PndPidCandidates. In case of just holding references to those (abstract) objects, nothing at all

has to be changed in the PndPid* classes.
Another improtant point is the access to the informations of the track objects. At the stage where you create the PndPidCandidate it is still not clear whether it fulfills the requirements for a specific particle type like electron, pion, etc. Consequently you have to "hard copy" all relevant properties
for all possible particle types of this track object (i.e. 5 time covariant matrices, 5x momentum, 5x vertex, etc.).
In case of holding a reference to the track object all infos are automatically available in your candidate. One has access to all public methods of this object and therefore also to the covariant matrices related to the different particle types. Another point is that the reference to the track object would allow to refit the track in the analysis part.

Best regards,
Bertram.
