II  Reference Manual  17
The femtoscopy analysis effort in ALICE is in the Physics Working Group 2. Thus, the software described here may be found in the PWG2/FEMTOSCOPY area when AliRoot is checked out.

The files (.tex, .pdf, etc.) for this documentation are cvs-controlled along with the AliFemto code in the ALICE repository under the PWG2/FEMTOSCOPY/Documentation area. While earth-moving changes in the framework are not anticipated, this manuscript will be to some extent a “living document.” The repository version of the documentation should be considered the “official” source of information on the package at any given time.

This document consists of two parts. The Users’ Guide introduces the package and, perhaps most interestingly to the reader, provides a blow-by-blow example of its use. The Reference Manual is essentially a listing and short description of the classes and the limited inheritance scheme.

AliFemto will often be run in a larger framework than a simple ROOT or AliRoot session, e.g. AliEn and the Task-Train framework in ALICE. This manual focusses only on AliFemto itself. For tutorials on how to run AliFemto within these contexts, see the ALICE femtoscopy web pages http://aliceinfo.cern.ch/Collaboration/PhysicsWorkingGroups/PWG2/Femtoscopy/.
Part I

Users’ Guide
1 Introduction - Basics and Motivation

Femtoscopy— the measurement of the space-time structure of dynamic systems at the fermi scale— is an integral tool in studies in high-energy particle (e.g. p+p) and heavy ion (e.g. Pb+Pb) collisions. It can also be a non-trivial, somewhat subtle tool, with nonobvious experimental “traps” which are periodically rediscovered as expertise evaporates and algorithms are lost.

Especially in large modern high energy experimental collaborations, complex experimental issues impact on this already-delicate tool. Furthermore, by the nature of such collaborations, “Physics Working Groups” (PWGs) are commonly formed, in which several collaborators work on similar physics topics (e.g. femtoscopy) which share common techniques and problems. Sharing of experience and solutions among PWG members is invaluable to work through problems quickly and to assure quality and consistency in physics results. In addition to regular discussions by phone/vrvs/email, sharing a common software analysis infrastructure allows for rapid and collaborative development, testing and sharing of solutions. Production of quality physics results in a timely manner demands the use of all available collaborative tools; while cross-checks are always crucial to an analysis, re-creating the many aspects of a wheel is sometimes a (all-too-common) waste of valuable manpower.

Just as code-sharing within a collaboration or PWG is desirable when analysis techniques are similar, code-sharing between collaborations or PWGs may be equally beneficial, if the similarities are sufficiently great. Code reusability is often claimed as one of the most important benefits of well-designed object-oriented programming. Through the development of standardized tools and inheritance schemes, high-energy physics has largely moved away from perpetual re-implementation of established algorithms. Previously, the student who needed a particular “twist” to, say a resonance-finding technique, would often find it easiest to start from scratch in a self-contained Fortran code. This was due to the fact that the previous student’s Fortran code lacked extensibility; e.g. interfaces and common blocks were specialized for a particular, narrow purpose. With care, languages such as C++ provide a natural solution. The student can focus on new aspects of her problem (the purpose, after all, of research) and the science behind it. The same objects and elements of the same code, developed and refined by others, are at her disposal; likewise, she will make her own contributions and everybody benefits. Likewise, so long as the detector and reconstruction configurations of two experiments are sufficiently similar, both collaborations may benefit by sharing some code.

This document discusses a two-particle correlation software package for use in the ALICE experiment at the LHC. It is based largely on the framework (StHbt) used since 1999 in the STAR experiment at RHIC, an experiment remarkably similar to ALICE in all respects. StHbt, itself, was developed by femtoscopy experts from earlier heavy ion experiments at the AGS and SPS; as such, it distills the generic features of any heavy ion femtoscopy analysis. The femtoscopy group in ALICE is arguably the most complete collection of the world’s experts in this sub-field ever assembled. It is hoped that this common analysis software will enable a maximum positive superposition of the experience of these experts.

1.1 Femtoscopy, HBT, and heavy ions

The feature distinguishing heavy ion from particle physics is the dominance of space-time geometry. This is manifest in the fact that we seek the geometrically-largest systems in order to approximate the infinite system in which thermodynamic variables and phases of matter become meaningful. At all stages of the the
dynamic system’s evolution, geometry rules. The geometric overlap anisotropy of the entrance channel is known to dominate the subsequent evolution of the bulk, and focusing systematics of geometric entrance-channel quantities (e.g. reaction-plane, impact parameter) yields much more information than geometric averages over these quantities. In the intermediate stage of the collision, path-length considerations are crucial to determine the physics of so-called “jet quenching” at the highest energies. Further, we seek a system in which coloured degrees of freedom are relevant over “large” length scales. Much of the dynamic bulk physics is reflected in the end freeze-out stage in collective observables (e.g. flow) which are usually defined in terms of space-momentum correlations. Clearly, geometry is a key defining feature of our field; momentum space alone is less than half the story.

However, particle momentum is precisely what we measure. Geometrical information must be inferred. The most direct and common method of doing so is through femtoscopy, the use of two-particle momentum-space correlations to probe fermi-scale emission zones. Experimental and theoretical aspects of femtoscopy are discussed at length elsewhere (2, and references within). Without becoming mathematical, the main point is to measure the increase (or decrease) of the likelihood of measuring a particle with a particular momentum, given the presence of another particle; in other words, the effect on the conditional probability. The effect to be measured is driven by the two-particle wavefunction, which depends on relative momentum (measured) and relative position (inferred). The probability \( A \) as a function of a measured relative quantity (typically relative momentum \( q \), so \( A \sim dN/dq \)) is usually dominated by detector acceptance and single-particle phasespace; the modification due to two-particle effects represent only a small perturbation. Thus, some sort of comparison to a reference distribution \( B(q) \) is usually performed. Ideally, \( B \) contains all single- and two-particle acceptance and efficiency effects and lacks only the sought-for correlation. The distribution \( B \) is often generated by so-called “mixed-event” techniques, and the correlation function \( C \), ideally containing only 2-particle correlations due to the relative wavefunction, given by

\[
C(q) = \frac{A(q)}{B(q)}
\]

It should be stressed, however, that neither using \( q = p_1 - p_2 \) as a two-particle variable, generating histograms/distributions \( A \) or \( B \), nor taking any ratio \( C \) must be associated with a correlation analysis, in general. See Section 1.2 below.

Briefly, some terminology which the reader may encounter. Two-particle femtoscopic measurements are related to the pioneering work of Hanbury-Brown and Twiss over half a century ago, to measure the angular size of stars (3). (The relationship, however, is somewhat more oblique than often realized.) Thus, similar analyses in high-energy physics are often referred to as “HBT” studies. For reference, the first actual application to high-energy physics was performed by Goldhaber, Goldhaber, Lee and Pais (4) shortly thereafter; thus correlations for pions reflect the “GGLP effect.” The rubrik of femtoscopy (5) is nowadays used in general.

1.2 The bones of a femtoscopic analysis

Similar algorithmic requirements and characteristics appear in a wide range of femtoscopic (and non-femtoscopic) analyses. AliFemto was designed as a common analysis framework for collaborators conducting diverse analyses sharing nevertheless a large overlap of techniques.

The design was driven by asking two questions: “What is a femtoscopy-style analysis, in general?” and
“What sorts of actions will be common to most analyses and what sorts will be person-specific?”

The first question may be answered with the following rough procedure. Names of class types inside square brackets \([\)]\) are discussed in Sections 2 and 3.1.

1. Obtain an event (usually, data associated with one collision) from somewhere. [AliFemtoEventReader]
2. (Optional) Write the event (or portions thereof) to a file. [AliFemtoEventReader]
   The output file does not necessarily use the same format as the input.
3. Decide to use or discard (“cut on”) the event in the analysis. [AliFemtoEventCut]
4. Select (“cut on”) the particles of interest. [AliFemtoParticleCut]
5. Form pairs of particles coming from the present event. [AliFemtoAnalysis]
6. Cut on these pairs. [AliFemtoPairCut]
7. Do “something” with these pairs. [AliFemtoCorrFctn]
   Usually, but not necessarily, this involves calculation of some relative variable (e.g. a relative momentum) and incrementing a histogram.
8. Usually, form other pairs of particles to construct a reference pair distribution. [AliFemtoAnalysis]
   Usually this is related to generating pairs (“mixed” pairs) of particles between the present event and similar events which are sitting in the EventBuffer.
9. Cut on these pairs. [AliFemtoPairCut]
   Almost always, it is an identical cut as used in step 6.
10. Do “something” with these pairs. [AliFemtoCorrFctn]
    Usually, but not necessarily, this involves calculation of some relative variable (usually the same one as in step 7) and incrementing a histogram.
11. Store the present event into the EventBuffer. [AliFemtoAnalysis]
12. Return to step 1 for the next event. [AliFemtoManager]

The first question is thus addressed by implementing the above basic functionality in methods of common classes. Considerations about the second question are reflected in the class inheritance structure and division of classes into “central” and User classes. We discuss this below.

2 The Structure and use of AliFemto

AliFemto is a flexible and extendable software package in the ROOT framework for performing two-particle femtoscopic studies. The basic design and structure of the package is essentially unchanged since its original deployment in the STAR Experiment at RHIC in 1999. However, its functionality and features have been developed considerably by continuous use in experimental and model analyses by the STAR-HBT group since then. The package was designed from the beginning to be independent of the STAR
2.1 Top level

The top-level structure of AliFemto is shown in Figure 1 in simplified Unified Modeling Language (UML) format. Here, we describe generally the classes shown, and their interaction. See the Reference Manual for details.

2.1.1 AliFemtoManager

There is a single AliFemtoManager object for any AliFemto session. AliFemtoManager controls all AliFemto actions; it is this (through the four methods Init(), ProcessEvent(), Report(), and Finish())...
2 THE STRUCTURE AND USE OF ALIFEMTO

2.1 Top level

ish()) with which the user interfaces. There is only one AliFemtoManager, instantiated by the user (or the AliFemtoTask– see beginning of Section 2), and objects are “plugged into” it by the user at runtime; see Section 2.4 for an example.

In order to proceed with an AliFemto study, the AliFemtoManager must have an AliFemtoEventReader, which passes event information to it. Optionally, the AliFemtoManager may also have one or more other AliFemtoEventReader objects, which are in “Write” mode and which takes the event data and writes it to a file. This is useful in order to change data format or to write out only a selection of the events or information within events. Also optionally (but usually the case), the AliFemtoManager will have one or more AliFemtoAnalysis objects; it is these which perform correlation studies. In principle, the AliFemtoManager may have neither Writers or Analyses, but in this case nothing gets done with the data.

2.1.2 AliFemtoEventReader

Typical users will not need to write AliFemtoEventReaders, but will instead use a standard one. We discuss its basics here just for reference.

The job of the AliFemtoEventReader is to pass events, upon request, to the AliFemtoManager. How the events are obtained (reading from a data or simulation file, reading from some location in memory filled by someone else, random generation within the AliFemtoReader itself...) is immaterial. Regardless of the data format read in by the AliFemtoReader, the data is passed to the AliFemtoManager in the form of an AliFemtoEvent. (See the Reference Manual for full details on the content and structure of AliFemtoEvent.) In this way, all of AliFemto code is unaware of external data sources or formats; all such dependency is confined to the AliFemtoEventReader-derived classes.

AliFemtoEventReader is, itself, a base class, with a pure virtual method ReturnHbtEvent(). For any given application/datasource (e.g. ALICE ESDs, Geant kine banks, RQMD files), a class must be written which inherits from AliFemtoEventReader. In the similar STAR package, of order 10 specific Reader classes are available for use now; they will be developed for ALICE as needed. It is expected that the typical user will not need to write his own AliFemtoEventReader class, but simply select one already written. It is one of these derived classes which is “plugged into” the AliFemtoManager. Once the user instantiates and configures the Reader, she typically does not interact with it anymore. A specific example is given in Section 2.4

2.1.3 AliFemtoAnalysis

Typical users will not need to write AliFemtoAnalysis objects, but will instead select among standard ones. We discuss the basics here just for reference.

The most important element of an AliFemto study is the Analysis. All Analysis classes derive from the interface class AliFemtoAnalysis, which has three important pure virtual methods. They must implement (even if it is a “do-nothing” method) a Finish() method, which will be invoked by the AliFemtoManager just before the session ends. Also, each AliFemtoAnalysis class must generate a Report() (c.f. Section 2.3).

Finally and most importantly, an AliFemtoAnalysis class must be able to Process() an AliFemtoEvent.
What it does with the AliFemtoEvent, even if it simply ignores it, is of course not important to the AliFemtoManager. In practice, naturally, an Analysis does process the data, making cuts and extracting correlations. The simplest AliFemtoAnalysis-derived class, AliFemtoSimpleAnalysis, is a good example of this. We discuss this in Section 2.2.

2.1.4 Action flow

Briefly, each time AliFemtoManager::ProcessEvent() is invoked by the user (or Task, or whatever), it obtains an AliFemtoEvent from its AliFemtoEventReader. It then passes this AliFemtoEvent to each of its Writers through the WriteHbtEvent() method. Finally, it passes the AliFemtoEvent to each of its AliFemtoAnalysis::ProcessEvent(AliFemtoEvent*).

2.2 Analysis level

The AliFemtoAnalysis is usually the focus of any AliFemto session. As discussed in Section 2.1.3, the only requirement of such an object in principle is that it must accept an AliFemtoEvent object via its ProcessEvent method. However, in practice, AliFemtoAnalysis-derived classes almost always contain Cuts, CorrFctns, etc. We describe these here.

A UML representation of the class structure of an Analysis is shown in Figure 2.

2.2.1 AliFemtoEventCut

The first action of an AliFemtoAnalysis is usually to invoke the method AliFemtoEventCut::Pass(AliFemtoEvent*). This returns a boolean value (and may, internally, store information about the AliFemtoEvent passed to it). If the event does not Pass, no further processing is done by the AliFemtoAnalysis—control returns to the AliFemtoManager.

The Reference Manual gives the full implementation of a simple AliFemtoEventCut.

2.2.2 AliFemtoParticleCuts

Depending on the topological nature of the particles being selected, one uses the class AliFemtoTrackCut, AliFemtoV0Cut, AliFemtoKinkCut, or AliFemtoXiCut. All of these inherit from AliFemtoParticleCut; see Figure 2. As with all cuts, these classes have pure virtual Pass() methods. An Analysis has two AliFemtoParticle cuts, corresponding to the two particles used in the correlation analysis.

There is “special” behaviour when the AliFemtoParticle cuts are applied. For each AliFemtoTrack/V0/Kink/Xi which Passes the cuts, an AliFemtoParticle is created. (The AliFemtoParticle objects themselves are created and used within the Analysis—the user is not concerned with them.) It is the AliFemtoParticle objects (not AliFemtoTracks etc) which are used at further steps in the Analysis for the current event. Important: all AliFemtoParticleCut-derived objects set the mass of the particle it selects. Based on kinematic and PID information, the user, through the AliFemtoParticleCut, decides e.g. that the Track is a proton; the user needs to tell the AliFemtoParticleCut the mass of the proton; c.f. Section 2.4.

When the AliFemtoParticle
2.2 Analysis level

```
AliFemtoAnalysis
    Report() = 0
    ProcessEvent(AliFemtoEvent*) = 0
    Finish() = 0

AliFemtoSimpleAnalysis
    mNumEventsToMix
    EventBegin()
    EventEnd()

AliFemtoEventCut
    Pass(AliFemtoEvent*) = 0
    Report() = 0

AliFemtoTrackCut
    Pass(AliFemtoTrack*) = 0

AliFemtoV0Cut
    Pass(AliFemtoV0*) = 0

AliFemtoKinkCut
    Pass(AliFemtoKink*) = 0

AliFemtoXiCut
    Pass(AliFemtoXi*) = 0

AliFemtoPairCut
    Pass(AliFemtoPair*) = 0
    Report() = 0

AliFemtoCorrFctn
    AddRealPair(AliFemtoPair*)
    AddMixedPair(AliFemtoPair*)
    Report() = 0
    Finish() = 0

AliFemtoPicoEvent

AliFemtoParticle
    <--many experimental quantities-->

AliFemtoCorrFctn
```

Figure 2: The basics of an Analysis configuration in UML representation. Most classes shown are base classes. Appending “= 0” to a method name denotes pure virtuality—the method is not defined in the base class, but must be defined in any instantiated class which derives from it. Yellow classes at the periphery are bases for user-written classes.

corresponding to this AliFemtoTrack is created, it is at this point that the mass of the proton is assigned to the particle.

2.2.3 AliFemtoPairCut

As the name suggests, user-written classes which derive from AliFemtoPairCut must have a Pass() method selecting AliFemtoPairs for further processing. (AliFemtoPairs are created and used internally; the user does not interact with them.) These cuts may, for example, try to discriminate “fake” pairs caused by splitting or (for the reference pair distribution) those tracks which would merge.

Importantly, the Analysis will automatically apply the same AliFemtoPair cut to pairs generated from “real” and from “mixed” events. Almost always, this is very important for femtoscopic analyses. However, if necessary, one may circumvent this behaviour by attaching AliFemtoPairCut objects to the correlation function object itself. In this case, different PairCuts may be applied to “real” and reference distributions.
2.3 Reports

2.3.4 AliFemtoCorrFctn

The “end result” of most femtoscopic studies is the correlation function. These compare somehow (often via a ratio of distributions, c.f. Equation 1) “real” and reference pairs. In general, then, a user-written class which derives from AliFemtoCorrFctn should implement AddRealPair(AliFemtoPair) and AddMixedPair(AliFemtoPair) methods. These methods may do whatever the user wishes, of course.

Most CorrFctn classes are rather simple. See the Reference Manual for a full listing of an example.

Often, one wishes to construct several correlation functions simultaneously (e.g. one in $Q_{inv}$, $q_{D}$, $\Theta_{opening}$ etc). For this reason, every AliFemtoAnalysis may have several AliFemtoCorrFctn objects. The same AliFemtoPairs are sent to each AliFemtoCorrFctn.

2.2.5 Action flow

The most important action of each AliFemtoAnalysis is in its ProcessEvent(AliFemtoEvent*) method. corresponds approximately to steps 3-11 of the list in Section 1.2. Upon being given an Event by the Manager, it first sends it to its EventCut. If AliFemtoEventCut::Pass(AliFemtoEvent*) returns false, the method returns with no further action.

Otherwise, an AliFemtoPicoEvent (essentially two lists of AliFemtoParticles, c.f. Section 2.2.2) is formed from the particles which Pass the AliFemtoParticleCuts. All possible pairs of these Particles are tested by the AliFemtoPairCut::Pass() method, and those which Pass are sent to each of the AliFemtoCorrFctn’s AddRealPair(AliFemtoPair*) methods.

Then, “mixed” AliFemtoPairs are formed by combining all AliFemtoParticles from the present event with those of previously-processed events, which have been stored in a collection of AliFemtoPicoEvents. (C.f. Figure 2; the user need not interact with this aspect of the code.) All such Pairs are formed. These are evaluated by the AliFemtoPairCut and those which Pass are sent to each of the AliFemtoCorrFctn::AddMixedPair(AliFemtoPair*) methods.

Finally, the AliFemtoPicoEvent is put into the list (c.f. Figure 2) of such objects for mixing with future events, and control returns to the AliFemtoManager.

As mentioned previously, the above procedure is just one typical of Analyses. In principle, one’s specific Analysis class might email the user each pair, or each particle at random.

2.3 Reports

Note that most classes above have Report() methods. These are simple user-written methods returning strings which can tell something about what happened to that class during the study. E.g. a AliFemtoEventCut might Report on how many events passed/failed the cut. The content of the Report is up to the user; it might be even an empty string or your spouse’s name.

At the end of the session, the Reports of the various objects are concatenated in the following way.

Each AliFemtoAnalysis collects Reports from its AliFemtoEventCut, AliFemtoParticleCut (for both the first and second particle), AliFemtoPairCut, and all of its AliFemtoCorrFctn objects. These are concatenate-
2.4 Example macro

Here, we show a specific macro which may be used to perform two Analyses which produce several CorrFctns. The example shown is for use in "pure" root. It may also be used as a simple macro in AliRoot or, as been previously mentioned, it is straightforward to wrap it in a Task or Train (or, in STAR, the BFC Chain).

Figures 3-7 in this Section are only cartoons suggesting what the code in the macro does— they are not UML. All classes in this specific example are user-written classes, with the exception of AliFemtoManager. (Several of them such as the AliFemtoEventReaderESD class, have already been written and are provided in the StHbt checkout.) For details, see Sections 2.1 and 2.2.

2.4.1 Initialization and plugging in the Reader

Figure 3 shows the beginning of the macro. As usual in a root macro, libraries are loaded first. For the femtoscopic analysis itself, only the AliFemto and AliFemtoUser libraries need be loaded. All internal AliFemto classes such as AliFemtoPair are found in the AliFemto directory. That directory also has a
2.4 Example macro

//-- now set up an Analysis
AliFemtoVertexAnalysis *anal = 
    new AliFemtoVertexAnalysis(3,-20.0,20.0);

// 1) The cuts...
// a) EventCuts - instantiate+configure+plug
AliFemtoBasicEventCut* evcut = new AliFemtoBasicEventCut();
evcut->SetEventMult(0,1000);
evcut->SetVertZPos(-20.0,20.0);
anal->SetEventCut(evcut);
// b) Track/V0/Kink/XG Cuts - instantiate+configure+plug
AliFemtoBasicTrackCut* trkcut = new AliFemtoBasicTrackCut();
trkcut->SetNHits(0,400); trkcut->SetPt(0.0,1.0);
trkcut->SetRapidity(-1.0,1.0); trkcut->SetDCA(-1.0,5.0); trkcut->SetCharge(1);
trkcut->SetMass(0.138);
anal->SetFirstParticleCut(trkcut);
anal->SetSecondParticleCut(trkcut);
// c) PairCuts - instantiate+configure+plug
AliFemtoDummyPairCut* prcut = new AliFemtoDummyPairCut();
anal->SetPairCut(prcut);

// 2) Now the Correlation Functions - instantiate+configure+plug
AliFemtoQinvCorrFctn* QinvCF =
    new AliFemtoQinvCorrFctn("Qinv",20,0.0,0.2);
anal->AddCorrFctn(QinvCF);
OpeningAngleCorrFctn* AngCF =
    new OpeningAngleCorrFctn("OpenAngVsQinv",5.0,0.0,1.0,0.0,25.0);
anal->AddCorrFctn(AngCF);
AliFemtoBPLCMSFrame3DCorrFctn* OslCF =
    new AliFemtoBPLCMSFrame3DCorrFctn("OutSideLong",20.0,0.0,0.1);
anal->AddCorrFctn(OslCF);

Figure 4: The construction of a specific AliFemtoAnalysis, including its cuts and collection of three CorrFctn objects. The dark shaded region in the cartoon denotes a collection.

A few simple CorrFcn, Reader, and Cut classes. The user will probably want to start with these and elaborate upon them; in this case, her new classes should go into the AliFemtoUser area.

In this example we use the specific AliFemtoEventReader-derived class AliFemtoEventReaderESD; not surprisingly, this Reader needs the ESD library loaded as well.

Since all Readers are user-written, their configuration methods will be specific to the class used. This is as it should be, since attempts to “foresee” all possible uses of the Reader classes will ultimately result in limitations later on, and sloppy work-arounds.

The AliFemtoManager is instantiated; note that its pointer is declared outside the scope of the macro, at the top. Finally, the AliFemtoEventReaderESD object is “plugged in” to the AliFemtoManager. This is indicated by the arrow in the cartoon; recall that this is not a UML diagram.

2.4.2 Adding Analyses

In Figure 4 we see construction of a specific Analysis (c.f. Section 2.2). An AliFemtoAnalysis-derived class, AliFemtoVertexAnalysis, is instantiated. (For reference, this class takes care to “mix” only those
2.4 Example macro

// 3) Final detail and then give the Analysis to the Manager
anal->SetNumEventsToMix(2);
TheManager->AddAnalysis(anal);

Figure 5: The final step in configuring the first Analysis is to set the number of events to mix when constructing the “background” distribution. This finished Analysis (dark grey box) is then added to the collection of Analyses for the AliFemtoManager (large light grey box), thus making the connection between these objects.

events close to each other in primary vertex position; it is very commonly used.) All configuration (vertex range, number of bins) takes place in the constructor, in this specific class.

The Cuts are (i) instantiated, (ii) configured, and (iii) “plugged into” this Analysis. Finally, three correlation functions are instantiated, configured, and added to the collection of CorrFctns for this analysis. (The collection is suggested by the dark-shaded box in the cartoon.) We note that one of the CorrFctn classes there is OpeningAngleCorrFctn. The fact that its name does not begin with “AliFemto” is a clue that this class is user-written, for her own purposes; it would not be included in ALICE nightly builds and would sit in the AliFemtoUser area.

Two notes: Firstly, we see that the same AliFemtoTrackCut is used for both the first and second particle—this is an analysis of identical pions. Secondly, as mentioned in Section 2.2.2, all AliFemtoParticleCuts must define a particle mass. This is not special to this specific example.

In Figure 4, the conglomeration of Analysis-related objects is not related to the AliFemtoManager instanti-
2.4 Example macro

// A reactionplane-sensitive analysis explicitly asking only for short tracks
AliFemtoRPAnalysis *analShort =
    new AliFemtoRPAnalysis(3,-100.0,100.0);

// Cuts
// 1a) the same EventCut as previous Analysis
mikesEventCut* ev2 = new mikesEventCut();
analShort->SetEventCut(ev2);
// 1b) differently-configured TrackCuts
dummyTrackCut* trkcutShort = new dummyTrackCut();
trkcutShort->SetNHits(0,50); trkcutShort->SetP(0.0,800.0);
trkcutShort->SetRapidity(-1.0,1.0);
trkcutShort->SetDCA(-1.0,5.0); trkcutShort->SetCharge(0);
trkcutShort->SetMass(0.138);
analShort->SetFirstParticleCut(trkcutShort);
analShort->SetSecondParticleCut(trkcutShort);
// 1c) the same Paircuts as previous Analysis
analShort->SetPairCut(prcut);
// 2) Now the CorrFctns (only 2 this time)
KstarCorrFctn* QinvCF2 = new KstarCorrFctn("Qinv2",20,0.0,0.2);
OpeningAngleCorrFctn* AngCF2 =
    new OpeningAngleCorrFctn();
analShort->AddCorrFctn(QinvCF2)
anal->AddCorrFctn(AngCF2);
// 3) Final detail and then give the Analysis to the Manager
analShort->SetNumEventsToMix(2);
TheManager->AddAnalysis(analShort);

cout << "Analysis initialization complete" << endl

Figure 6: A second Analysis is instantiated, configured, and added to the collection. The same procedure is followed as for the first Analysis (Figures 4 and 5), though the two Analyses know nothing about each other.

ated in the previous Figure. In Figure 5, the connection is made. Firstly, the “final detail” for the Analysis is completed– the number of events to mix for the reference distribution is set, and the now-fully-configured Analysis is added to the collection of Analyses (denoted by the large grey box).

The structure is now complete in principle– a useful correlation study may proceed. We may add one or more completely separate Analyses, if we wish. This is shown in Figure 6. The only points here are that the second (and any subsequent) Analyses are set up similarly to the first one discussed above, and there is no connection the Analyses in the AliFemtoManager’s collection of Analyses.

2.4.3 Processing data

Finally, Figure 7 moves beyond construction of the collection of objects, and commands a processing of the data. We note that all “external” interaction is with the AliFemtoManager class. If the Reader needs to interact with the “outside world” (e.g. by opening a file or pointing to a location in memory), then it is up to that specific class, using its own specific methods, to take care of that.
3 Code Organization

3.1 Directory structure: core and user classes

The femtoscopy analysis effort in ALICE falls in the soft physics working group (PWG2). Thus, when checking out the full AliRoot from StHbt, it will be found in the PWG2/FEMTOSCOPY/ area. (One may also check out only PWG2 files using
cvs -qz2 -d :pserver:cvs@alisoft.cern.ch:/soft/cvsroot co PWG2.
This is useful if running AliFemto in “standalone” mode.)
In this directory are found two subdirectories. The first is `AliFemto/`, which holds about 80 classes (as of July 10, 2007), such as `AliFemtoManager`, `AliFemtoPair`, etc. It also holds all of the base classes for user-derived code (e.g. `AliFemtoCorrFctn`) and one or two simple examples of classes which derive from these (e.g. `AliFemtoQinvCorrFctn`). These last might be useful templates for users writing more sophisticated Cuts, CorrFctns, Readers, Analyses, etc. The `AliFemto/` subdirectory is included in the official nightly build. Files there must obey ALICE coding standards, and limited `StHbt` access is anticipated.

The second directory is `AliFemtoUser/`. Files in this subdirectory are not included in the nightly build. This area is meant to be a repository for user code, typically Cuts, CorrFctns, etc, which might be of interest to others working on some analysis. It is `StHbt` archived, and widespread read/write `StHbt` access is anticipated.

Seperate shared object (.so) libraries are built for the `AliFemto/` and `AliFemtoUser/` area; c.f. the macro snip in Figure 3.

## 4 Known problems

As of July 10, 2007, there are no known problems with `AliFemto`. However, this Users’ Guide is incomplete, in that it does not discuss so-called “theoretical” correlation studies covered in the `AliFemto-Model` classes. This will be remedied in the next version of the Users’ Guide. In the meantime, see an excellent tutorial on the subject on the ALICE femtoscopy webpage [http://aliceinfo.cern.ch/Collaboration/PhysicsWorkingGroups/PWG2/Femtoscopy/](http://aliceinfo.cern.ch/Collaboration/PhysicsWorkingGroups/PWG2/Femtoscopy/).
Part II

Reference Manual

The Reference Manual is being finalized.
References


[5] Lednicky, R., nucl-th/0212089