
Revision 0.5

June 1, 2006
Contents

0 About this document .................................................. 1

I Users’ Guide .................................................................. 2

1 Introduction ................................................................... 3
  1.1 Femtoscopy, HBT, and heavy ions ................................. 3
  1.2 The bones of a femtoscopic analysis ......................... 4

2 The Structure and use of StHbt ....................................... 5
  2.1 Top level .................................................................. 5
    2.1.1 StHbtManager .................................................. 6
    2.1.2 StHbtEventReader ............................................. 6
    2.1.3 StHbtBaseAnalysis ............................................ 7
    2.1.4 Action flow ...................................................... 7
  2.2 Analysis level .......................................................... 7
    2.2.1 StHbtEventCut .................................................. 7
    2.2.2 StHbtParticleCuts .............................................. 8
    2.2.3 StHbtPairCut .................................................... 9
    2.2.4 StHbtCorrFctn .................................................. 9
    2.2.5 Action flow ...................................................... 9
  2.3 Reports ................................................................ 9
  2.4 Example macro ....................................................... 10
    2.4.1 Initialization ...................................................... 10
    2.4.2 Adding Analyses ............................................... 12
    2.4.3 Processing data ................................................ 14

3 Code Organization .......................................................... 15
  3.1 Core and user classes and directories ....................... 15
  3.2 STAR Class Library ............................................... 15
  3.3 Management and evolution paths ............................. 16
4 Range of applicability
  4.1 Platforms and compilers ................................. 16
  4.2 Frameworks ............................................. 16

5 Known problems .............................................. 16

II Reference Manual ............................................. 17

6 Base Classes ................................................... 18
  6.1 StHbtBaseAnalysis ........................................ 20
  6.2 StHbtCorrFctn .............................................. 20
  6.3 StHbtCutMonitor ........................................... 20
  6.4 StHbtEventReader ........................................... 20
  6.5 StHbtEventWriter .......................................... 20
  6.6 StHbtHiddenInfo ........................................... 20
  6.7 StHbtLikeSignCorrFctn ................................... 20
  6.8 Cut classes ................................................ 20
    6.8.1 StHbtEventCut ......................................... 20
    6.8.2 StHbtKinkCut .......................................... 20
    6.8.3 StHbtPairCut .......................................... 20
    6.8.4 StHbtParticleCut ...................................... 20
    6.8.5 StHbtTrackCut ......................................... 20
    6.8.6 StHbtV0Cut ............................................. 20
    6.8.7 StHbtXiCut ............................................. 20

7 Infrastructure Classes ........................................ 20
  7.1 StHbtAnalysis ............................................. 20
  7.2 StHbtAnalysisCollection ................................ 20
  7.3 StHbtCorrFctnCollection ................................. 20
  7.4 StHbtCoulomb ............................................. 20
  7.5 StHbtCutMonitorCollection ............................... 20
  7.6 StHbtCutMonitorHandler ................................. 20
7.7 StHbtEnumeration ............................................. 20
7.8 StHbtEvent ..................................................... 20
7.9 StHbtEventWriterCollection ................................. 20
7.10 StHbtHelix ..................................................... 20
7.11 StHbtKink ...................................................... 20
7.12 StHbtKinkCollection .......................................... 20
7.13 StHbtLikeSignAnalysis ....................................... 20
7.14 StHbtLinkDef .................................................. 20
7.15 StHbtManager .................................................. 20
7.16 StHbtPair ...................................................... 20
7.17 StHbtParticle .................................................. 20
7.18 StHbtParticleCollection ...................................... 20
7.19 StHbtPicoEvent ............................................... 20
7.20 StHbtPicoEventCollection .................................... 20
7.21 StHbtPicoEventCollectionVector ............................ 20
7.22 StHbtPicoEventCollectionVectorHideAway ................... 20
7.23 StHbtString .................................................... 20
7.24 StHbtTrack ..................................................... 20
7.25 StHbtTrackCollection ......................................... 20
7.26 StHbtTypes ..................................................... 20
7.27 StHbtV0 ......................................................... 20
7.28 StHbtV0Collection ............................................. 20
7.29 StHbtVector .................................................... 20
7.30 StHbtVertexAnalysis .......................................... 20
7.31 StHbtVertexMultAnalysis ..................................... 20
7.32 StHbtXi ........................................................ 20
7.33 StHbtXiCollection ............................................. 20

8 Examples of User classes ........................................ 20

A For STAR-HBT Users: Liberation from root4star ............... 20
B Acknowledgements
About this document

This is the Users’ Guide and Reference Manual for the StHbt software package, for use in the STAR and ALICE experiments. It is downloaded with the main code (.tex source, .eps figure files, and make file) and should be considered the “official” source of information on the package.

As of this version (v0.5), the code and documentation sit on a public, non-CVS-controlled website, http://www.physics.ohio-state.edu/HIRG/StHbt/. CVS commitment is envisioned soon.

This software is available free of charge, with no conditions, and with no promise of support in general. As much help as possible will be given to STAR and ALICE members however. For questions, write to Mike Lisa at lisa@mps.ohio-state.edu
Part I

Users’ Guide
1 INTRODUCTION

1 Introduction

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/vrsvs/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 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 developed by the STAR-HBT PWG over several years. It is applicable for model studies and for experimental analysis within the STAR experiment and experiments similar to STAR.

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 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
1.2 The bones of a femtoscopic analysis

It is 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 (1, 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 (1), 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 (2). Thus, similar analyses in high-energy physics are often referred to as “HBT” studies; this is reflected in the name “StHbt.” For reference, the first actual application to high-energy physics was performed by Goldhaber, Goldhaber, Lee and Pais (3) shortly thereafter; thus correlations for pions reflect the “GGLP effect.” The general rubrik (4) of femtoscopy 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. StHbt 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. [StHbtReader]
2 THE STRUCTURE AND USE OF STHBT

2. (Optional) Write the event (or portions thereof) to a file, [StHbtReader]
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. [StHbtEventCut]

4. Select (“cut on”) the particles of interest. [StHbtParticleCut]

5. Form pairs of particles coming from the present event. [StHbtAnalysis]

6. Cut on these pairs. [StHbtPairCut]

7. Do “something” with these pairs. [StHbtCorrFctn]
   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. [StHbtAnalysis]
   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. [StHbtPairCut]
   Almost always, it is an identical cut as used in step 6.

10. Do “something” with these pairs. [StHbtCorrFctn]
    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. [StHbtAnalysis]

12. Return to step 1 for the next event. [StHbtManager]

2 The Structure and use of StHbt

StHbt is a flexible and extendable software package in the ROOT 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. StHbt was designed from the beginning to be independent of the STAR analysis framework (root4star). Thus, it is easily used for ALICE-specific analyses (either in AliRoot or “vanilla” root) or for model studies in any root flavor.

In this Section, we discuss the structure of the code, and a specific example of how to use it.

2.1 Top level

The top-level structure of StHbt is shown in Figure 1 in UML format. Here, we describe generally the classes shown, and their interaction. See the Reference Manual for details.
2.1 Top level

2.1.1 StHbtManager

StHbtManager controls all StHbt actions; it is this (through the four methods Init(), ProcessEvent(), Report(), and Finish()) with which the user (or root4star or AliRoot) interfaces. There is only one StHbtManager, instantiated by the user, and objects are “plugged into” it by the user at runtime; see Section 2.4 for an example.

In order to proceed with an StHbt study, the StHbtManager must have an StHbtEventReader, which passes event information to it. Optionally, the StHbtManager may also have one or more other StHbtEventReader 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 StHbtManager will have one or more StHbtAnalysis objects; it is these which generate correlation studies. In principle, the StHbtManager may have neither Writers or Analyses, but in this case nothing gets done with the read data.

2.1.2 StHbtEventReader

The job of the StHbtEventReader is to pass events, upon request, to the StHbtManager. How the events are obtained (reading from a data or simulation file, reading from root4star memory, random generation within

Figure 1: The large-view structure of the code. Yellow classes at the periphery are bases for user-written classes.
the Reader itself) are immaterial. Regardless of the data format read in by the Reader, the data is passed to the StHbtManager in the form of an StHbtEvent. In this way, all of StHbt code is unaware of external data sources or formats; all such dependency is confined to the StHbtEventReader classes.

StHbtEventReader 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 StHbtEventReader; of order 10 specific Reader classes are available for use now. It is one of these derived classes which is “plugged into” the StHbtManager.

2.1.3 StHbtBaseAnalysis

The most important element of an StHbt study is the Analysis. All StHbtAnalysis classes derive from the interface class StHbtBaseAnalysis, 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 StHbtManager just before the session ends. Also, each StHbtAnalysis class must generate a Report (c.f. Section 2.3).

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

2.1.4 Action flow

Briefly, each time StHbtManager::ProcessEvent() is invoked (by the calling routine in root4star, AliRoot, or root macro), it obtains an StHbtEvent from its Reader. It then passes this StHbtEvent to each of its Writers through WriteHbtEvent method. Finally, it passes the StHbtEvent to each of its StHbtAnalyses.

2.2 Analysis level

The StHbtAnalysis is usually the focus of any StHbt session. As discussed in Section 2.1.3, the only requirement of such an object in principle is that it must accept an StHbtEvent object via its ProcessEvent method. However, in practice, StHbtBaseAnalysis-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 StHbtEventCut

The first action of a StHbtAnalysis is usually to invoke the method StHbtEventCut:Pass(StHbtEvent*). This returns a boolean value (and may, internally, store information about the StHbtEvent passed to it). If the event does not Pass, no further processing is done by the StHbtAnalysis– control returns to the StHbtManager.
2.2 Analysis level

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.

2.2.2 StHbtParticleCuts

Depending on the topological nature of the particles being selected, one uses the class StHbtTrackCut, StHbtV0Cut, StHbtKinkCut, or StHbtXiCut; see Figure 2. As with all cuts, these classes have pure virtual Pass() methods. An Analysis has two StHbtParticle cuts, corresponding to the two particles used in the correlation analysis.

There is "special" behaviour when the StHbtParticle cuts are applied. For each StHbtTrack/V0/Kink/Xi which Passes the cuts, an StHbtParticle is created. (StHbtParticle objects are created and used within the Analysis—the user is not concerned with them.) It is the StHbtParticle objects (not StHbtTracks etc) which are used at further steps in the Analysis for the current event. **Important:** all StHbtParticleCut-derived objects know the mass of the particle it selects. Based on kinematic and PID information, the user, through the StHbtParticleCut, decides e.g. that the Track is a proton; the user needs to tell the StHbtParticleCut the mass of the proton; c.f. Section 2.4. When the StHbtParticle corresponding to this Track is created, it is at this point that the mass of the proton is assigned to the particle.
2.2.3 StHbtPairCut

As the name suggests, user-written classes which derive from StHbtPairCut must have a Pass method selecting StHbtPairs for further processing. 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 StHbtPair 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 StHbtPairCut objects to the CorrFctn object itself. In this case, different PairCuts may be applied to “real” and reference distributions.

2.2.4 StHbtCorrFctn

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 StHbtCorrFctn should implement a AddRealPair and AddMixedPair method. These methods may do whatever the user wishes, of course.

Often, one wishes to construct several correlation functions simultaneously (e.g. one in $Q_{\text{inv}}$, $\vec{q}_{3D}$, $\Theta_{\text{opening}}$ etc). For this reason, every StHbtAnalysis may have several StHbtCorrFctn objects. The same StHbtPairs are sent to each StHbtCorrFctn.

2.2.5 Action flow

The most important action of each StHbtAnalysis is in its ProcessEvent 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 StHbtEventCut::Pass(StHbtEvent*) returns false, the method returns.

Otherwise, an StHbtPicoEvent (essentially two lists of StHbtParticles, c.f. Section 2.2.2) is formed from the particles which pass the StHbtParticleCuts. All possible pairs of these Particles tested by the StHbtPairCut::Pass() method, and those which pass are sent to each of the StHbtCorrFctn’s AddRealPair methods.

Finally, “mixed” StHbtPairs are formed by combining all StHbtParticles from the present event with those of previously-processed events, which have been stored in a collection of StHbtPico events. (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 StHbtPairCut and those which pass are sent to each of the StHbtCorrFctn::AddMixedPair methods.

Finally, the StHbtPicoEvent is put into the list of such objects for mixing with future events, and control returns to the StHbtManager.

2.3 Reports

Note that most classes above have Reports(). These are simple user-written strings which can tell something about what happened to that class during the study. E.g. a StHbtEventCut might Report on how many events passes/failed the cut. The content of the Report is up to the user; it might be even an empty string.
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 root4star or AliRoot. We will point out the differences when using the “maker” formalism in root4star.

2.4.1 Initialization

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 (StHbt/Base-derived) classes, with the exception
we also see the beginnings of the study structure; c.f. Section 2.1. The StHbtReader (Al-

musesEventCut evcut = new mikesEventCut(); evcut->SetEventMult(500,80000); evcut->SetVertexZPos(-20.0,20.0); anal->SetEventCut(evcut);

// b) Track/V0Kink/Xi Cuts - instantiate+configure+plug
  // a) EventCuts - instantiate+configure+plug
  mikesEventCut* evcut = new mikesEventCut();
  evcut->SetEventMult(500,80000); evcut->SetVertexZPos(-20.0,20.0);
  anal->SetEventCut(evcut);
  // b) Track/V0Kink/Xi Cuts - instantiate+configure+plug
  PIDTrackCut* tricut = new PIDTrackCut();
  tricut->SetNHits(0,400); tricut->SetIPT(0,0.8,0.0);
  tricut->SetRapidity(-1.0,1.0);
  tricut->SetDCA(-1.0,5.0); tricut->SetCharge(1);
  anal->SetFirstParticleCut(tricut);
  anal->SetSecondParticleCut(tricut);
  // c) PairCuts - instantiate+configure+plug
  AntiSplittingPairCut* prcut = new AntiSplittingPairCut();
  anal->SetPairCut(prcut);
  // 2) Now the Correlation Functions - instantiate+configure+plug
  QinvCorrFctn* QinvCF = new QinvCorrFctn("Qinv",20.0,0.0,2.2);
  anal->AddCorrFctn(QinvCF);
  OpeningAngleCorrFctn* AngCF =
    new OpeningAngleCorrFctn("OpeningAngle",5.0,0.0,1.10,0.0,25.0);
  anal->AddCorrFctn(AngCF);
  BPLCMSFrame3DCorrFctn* OsICF =
    new BPLCMSFrame3DCorrFctn("OutSideLong",20.0,0.0,1);
  anal->AddCorrFctn(OsICF);

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

of StHbtManager. For details, see Sections 2.1 and 2.2.

Figure 3 shows the beginning of the macro. Unless some special framework (e.g. root4star or AliRoot) dependent library is needed, only the StarClassLibrary, StHbt, and HbtUserCode libraries need be loaded. Again, we emphasize that any need for framework-specific libraries must come only because of the specific Reader used. The StHbtMuDSTReader, for example, reads STAR-DST data from memory (not from a file) in the root4star maker mode, and so needs STAR-specific libraries loaded.

StHbt also uses a few of the classes from the STAR Class Library (5), which, despite its name, is a stand-alone library not special to STAR. It includes objects like StPhysicalHelix, which is useful for rough track extrapolation.

In Figure 3 we also see the beginnings of the study structure; c.f. Section 2.1. The StHbtReader (Ali-
iStHbtEventReader in this case, a class which reads a TTree file from disk) is instantiated and configured (i.e. given a list of files to process). 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
2.4 Example macro

2.4.2 Adding Analyses

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

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 StHbtManager (large light grey box), thus making the connection between these objects.

classes will ultimately result in limitations later on, and sloppy work-arounds.

The StHbtManager is instantiated; note that its pointer is declared outside the scope of the macro, at the top. Finally, the AliStHbtEventReader object is “plugged in” to the StHbtManager. This is indicated by the arrow in the cartoon; recall that this is not a UML diagram.
A readionplane-sensitive analysis explicitly asking only for short tracks

```cpp
StHbtRPAnalysis *analShort = new StHbtRPAnalysis(3,100.0,100.0);
```

```cpp
// Cuts
// 1a) the same EventCut as previous Analysis
analShort->SetEventCut(eventcut);
// 1b) differently-configured TrackCuts
dummyTrackCut* trkcutShort = new dummyTrackCut();
trkcutShort->SetNHits(0,50); trkcutShort->SetPt(0.0,800.0);
trkcutShort->SetRapidity(-1.0,1.0);
trkcutShort->SetDCA(-1.0,5.0); trkcutShort->SetCharge(0);
trkcutShort->SetMass(0.138);
analShort->SetFireParticleCut(trkcutShort);
analShort->SetSecondParticleCut(trkcutShort);
// 1c) the same PairReus as previous Analysis
analShort->SetPairCut(paircut);
```

```
// 2) Now the CorrFctns (only 2 this time)
KstarCorrFctn* QinvCF2 = new
KstarCorrFctn("Qinv2",*0,0,0,0.2);
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);
```

```cpp
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 fist Analysis (Figures 4 and 5), though the two Analyses know nothing about each other.

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 that is that there is no connection the Analyses in the StHbtManager’s collection of Analyses.

Two notes: Firstly, we see that the same StHbtTrackCut 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 StHbtParticleCuts must define a particle mass. This is not special to this specific example.
2.4 Example macro

```cpp
// --------- process the events
if (TheManager->in())
    cout << "Problem? - non-zero initialization value\n";

int Status;
int nEventsProcessed = 0;
do {
    nEventsProcessed++;
    cout << " + EVENT " << nEventsProcessed << " +\n";
    Status = TheManager->ProcessEvent();
} while (Status && (nEventsProcessed < nEvents));
TheManager->Finish();
}
// end of macro
```

Figure 7: Construction of the StHbt study complete, event looping is trivial. Note that all interaction with the code is through only a few methods of the StHbtManager object.

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 StHbtManager 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.

In Figure 7, the processing is ordered within a “pure root” macro directly. This can be different in other frameworks. For example, in the STAR Maker schema (6), the StHbtManager Init(), ProcessEvent(), Report() and Finish() methods will be invoked by the StHbtMaker Init(), Make(), and Finish() methods. Indeed, the StHbtMaker does almost nothing else than perform these simple invocations.
3 Code Organization

3.1 Core and user classes and directories

As should be clear from the above, there is a natural distinction between classes common to all StHbt usages, and those specific to a given user/study. These we denote as “core” and “user” classes, respectively.

The core classes come with the standard StHbt download. There are about 35 such classes (some of them trivial collection classes); these are listed in the Reference Manual. Of these, about 15 are base classes only. These are indicated by yellow-shading in Figures 1 and 2. The user implements specific classes which inherit from these, in order to run an StHbt session and perform her study. These classes may be found in the Base/ subdirectory, and include the StHbtCorrFctn base class (c.f. Figure 2 and Section 6.2) from which QinvCorrFctn class (c.f. Figure 4) inherits. The remaining ∼ 20 classes are functional components of StHbt and include the StHbtManager, StHbtEvent and StHbtTrack classes. The typical user does not usually need to deal with these classes directly.

The user must write (or obtain from another user or a pool of classes) classes for Readers, CorrFctns, Cuts, and, in principle, Analysis (see below). These should be stored in a separate subdirectory structure and comprise a separate library from the “core” classes, as seen in the Load() arguments in Figure 3. A main point of the common interface (base) classes is that common functionality allows trivial sharing of “User” objects among analysers. Ideally, a class may be simply copied from another user and inserted with no change into another’s analysis. With StHbt, this benefit applies also between collaborators in STAR and ALICE.

Note that StHbtAnalysis classes are kept in Infrastructure, although they are kind of a User class. The reason is that these classes are added modified only very infrequently; the typical user simply selects one and uses it. In principle, the user might modify one or write his own. Thus the choice to store StHbtAnalysis classes in Infrastructure is somewhat arbitrary.

The suggested directory structure might be as follows

```
StHbt/
  Base/
  Infrastructure/
  Documentation/
  HbtUserCode
  CorrFctn/
  Cut/
  Reader/
  StarClassLibrary/
```

3.2 STAR Class Library

StHbt also uses a few classes from the STAR Class Library (5), which, despite its name, is a standalone library not special to STAR. It includes objects like StPhysicalHelix, which is useful for rough track ex-
3.3 Management and evolution paths

As has been emphasized, StHbt classes can be used in both STAR and ALICE. Thus, classes (especially User classes) are easily traded between collaborations as well as within one. At first, it might seem natural to manage one version of StHbt in a cvs (or similar) repository. However, this may prove burdensome. It makes possible the scenario that evolutionary changes made by STAR could interfere with ALICE physics analysis. If ALICE finds it beneficial to adopt/steal changes made by STAR, then the adoption is trivial. However, nothing of the sort should be forced. Pursuit of the goal of code reusability (or any other computing consideration) must not ever interfere with physics.

Thus, STAR’s version of StHbt will be kept in a separate cvs repository than that of ALICE, and the evolution of the codes will be independent by default. StHbt main structure and core classes have shown great historical stability; e.g. as of May 2006, STAR’s StHbtManager was last touched in 2001 and StHbtAnalysis in 2002. This suggests that, even without active oversight, compatibility between the StHbt version of ALICE and STAR will remain largely intact.

4 Range of applicability

4.1 Platforms and compilers

Just list to where it’s been applied so far. And say where it should work probably

4.2 Frameworks

Distinct from platforms, the three frameworks are root4star, AliRoot, and “just” root.

Also Blood types and reusability... might be interesting to use the blood types thingy

Discuss that interface to root4star / AliRoot / “pure” root is only in Reader classes and in the three “tenuous connection” calls (as discussed in my presentation to PW2).

5 Known problems

AFAIK none. But “watch this space.”
Part II

Reference Manual
6 Base Classes

In general...

These are usually bases of User classes, needed for interface. Note especially the pure virtual methods.
6.1 StHbtBaseAnalysis

6.2 StHbtCorrFctn

6.3 StHbtCutMonitor

6.4 StHbtEventReader

6.5 StHbtEventWriter

6.6 StHbtHiddenInfo

6.7 StHbtLikeSignCorrFctn

6.8 Cut classes

6.8.1 StHbtEventCut

6.8.2 StHbtKinkCut

6.8.3 StHbtPairCut

6.8.4 StHbtParticleCut

6.8.5 StHbtTrackCut

6.8.6 StHbtV0Cut

6.8.7 StHbtXiCut

7 Infrastructure Classes

7.1 StHbtAnalysis

7.2 StHbtAnalysisCollection

7.3 StHbtCorrFctnCollection

7.4 StHbtCoulomb

7.5 StHbtCutMonitorCollection

7.6 StHbtCutMonitorHandler

7.7 StHbtEnumeration

7.8 StHbtEvent

7.9 StHbtEventWriterCollection

7.10 StHbtHelix
REFERENCES

Include somehow the table I made for the ppt file as “backup” for the PW2 presentation.

Although

B Acknowledgements

This document was written approximately seven years after the first implementation of StHbt and its first test in STAR’s Mock Data Challenge (MDC3). In the intervening time, it has seen continuous use in STAR— for both “traditional HBT” and other (e.g. jet and resonance) studies. In the process, more sophisticated algorithms and features have been implemented by many collaborators. Among the more prominent developers of StHbt, I would like to thank Zbigniew Chajecki, Dominik Flierl, Adam Kisiel, Brian Laziuk, Mercedes López-Noriega, Gael Renault, Fabrice Retière, Randy Wells, and Robert Willson. Thanks most of all to Frank Laue, who led the design and implementation of the original version.

References


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


[6] Unfortunately, the best (!?!) documentation of the basic STAR Maker analysis framework is a brief presentation by Victor Perevoztchikov which may be found on the STAR computing tutorial website: http://www.star.bnl.gov/STAR/comp/train/tut/Maker-in-STAR/Victor-Makers.html