1st draft

Beta tutorial for 6.0.2

Natalia Kuznetsova

June 3, 1998

Abstract

This is a very brief UCSB-specific tutorial intended to get you started with the BaBar software. It is using release 6.0.2 as an example. We will start with considering what Beta is and what is necessary to run it here at UCSB. We will then construct a sample analysis module, addressing in the process such issues as handling lists and list iterators, booking and filling histograms, obtaining Monte Carlo truth table information, and generating the decay modes that one is interested in using EvtGen.

Contents

1 What is Beta?

There is some documentation about Beta on the Web at the following URL:

http://www.slac.stanford.edu/BFROOT/doc/www/Beta.html

To give a very brief summary, Beta is a package (actually, about 9 packages used together) that provides a set of analysis tools for BaBar. Its basic concepts are:

- BtaCandidate is Beta's name for a candidate particle in the event in question. For example, charged tracks can be candidate pions, and neutral clusters can be candidate gammas. To see what's available for a BtaCandidate, look at the following header file:

  /nfs/babar2/BFROOT/dist/releases/6.0.2/Beta/BtaCandidate.hh

  Essentially, one can get basic properties, such as mass, charge, four-momentum, etc.. There are also pointers to classes that allow one to access, say, particle ID information (e.g., class PidInfoSummary) or vertex information (class BtaAbsVertex). Finally, BtaCandidates maintain their genealogy information, so they know their mothers and daughters, if any. In section "Handling lists" I will consider some examples of how to get this information.

- Operators allow one to combine several candidates into a new one. There are a number of ways to do this: vertexing, simple 4-vector addition, mass constrained fitting, etc..

- Finders are basically analysis algorithms intended to do a specific task, such as finding a K^0 by vertexing pairs of pion candidates and applying basic cuts. Most of the available finders are in the BtaTools package.

- Associators serve to associate one candidate with another during analysis, e.g., a reco candidate with the corresponding Monte Carlo truth candidate. The class that's used here is calles, not surprisingly, BtaAssociator.

There is a Beta tutorial on the Web, at the following URL:

http://www.slac.stanford.edu/BFROOT/doc/www/Physics/BetaTutorial/BetaTutorial.html

It is rather SLAC-specific, while this tutorial is intended to be UCSB-specific, and (hopefully) a little more detailed.

2 Where to get information?

There are two main resources that I find very handy when I have questions:

- BaBar hypernews discussion groups on the Web at the following URL:

  http://babar-hn.slac.stanford.edu:5090/HyperNews/index

  The two groups I look at most often are "Preliminary/unconfirmed bugs, problems, frustrations, fixes" and "Analysis tools".

- What documentation there is in the release packages themselves. All the packages that we currently have here at UCSB are located at (on electron):

  /babar2/BFROOT/dist/releases/

  It has been my experience that header files (.hh) often provide especially useful information.

There are also various tutorials and write-ups on the Web, but since things change so quickly I find it much more convenient to use the above-mentioned resources.

3 SLAC UNIX account

Before you can do anything related to the BaBar software, you have to make sure your SLAC UNIX account is set up properly. If you don't have an account there, there are detailed instructions about how to get one on the Web, at the following URL:

http://www.slac.stanford.edu:/BFROOT/doc/Computing/NewUser/Account.html

Briefly, there are two steps: you need to make sure you are listed in the Babar Membership Database, and if you are, then you can apply for a SLAC UNIX account. After you get it, you can check that it works by logging into SLAC machines from here (I usually use morgan04.slac.stanford.edu). Once you have a SLAC account you should also get an AFS account. The safest thing to do is to log into any SLAC machine, type "movetoafs" and then log out and wait till the next day (the transfer to AFS usually happens automatically at midnight). After this, your home directory

at SLAC should be something like:
/afs/slac.stanford.edu/u/ec/your_user_name.

Now, at UCSB, if everything went ok, you should be able to type “klog” on electron (which is our AFS machine), type in your SLAC password when prompted for it, and get access to SLAC AFS system.

If you encounter problems at any stage of getting a SLAC account, the best person to contact (in my opinion) is Tom Glanzman (DRAGON@SLAC.Stanford.EDU).

4 What you need to change in your account set-up at UCSB

If you are sure you have a working SLAC UNIX account, the next step is to add a few things to your home directory here at UCSB. First, create a file name BFCURRENT in you home directory, and put the string “6.0.2” in it. Then edit your .cshrc file. I wrote a simple script that sets most of the BaBar environment variables that you need: it is in /homes/natalia, and is called BaBarUCSB.cshrc. I will try to keep it updated. You can just add “source /homes/natalia/BaBarUCSB.cshrc” at the end of your .cshrc file. Be aware that I assumed in that script that your user name at SLAC is the same as at UCSB (which it is for most people), so if it’s not, you should either copy BaBarUCSB.csrch to your area and modify it appropriately, or let me know and I will change it for you. The script should also set up Objectivity environment which you need for 6.0.2, so you don’t need to worry about it.

After this is done, source .cshrc, or (better) log out and log back in.

5 Setting up a test release

In order to create your own test release, login to electron, go to /babar2, and create your own directory there, if you don’t have one already. I will be referring to it as your_directory. Now, cd to your_directory. Assuming you are in /babar2/your_directory, do a “klog”, and then type

```
newrel -t 6.0.2 my6.0.2
```

Sometimes one gets error messages, such as “newrel: command not found” at this stage. If this happens, it might help to log out and log back in again, to make sure that .cshrc gets executed properly, and to type “rehash” to update the hash tables.

You can also try typing “echo $BFARCH”; it should be OSF1V4-native-Objy, and “echo $BFCURRENT”, which should be 6.0.2. If all the environmental variables are correct but you still can't get it to work, chances are there's something wrong with your SLAC account.

If everything works, the “newrel” command should create a directory named “my6.0.2”, with some initial packages there (such as man, results, test, and so on).

Now, cd to my6.0.2, and type:

```
addpkg Beta V00-06-00
```

This should add a *package* named “Beta” to your my6.0.2. “V00-06-00” is a particular *tag* or version for this package. It is very important to get the rights versions of packages, because only certain tags actually work (and finding out which do isn't always easy!).

Now add a few more packages to my6.0.2:

```
addpkg BetaUser V00-06-00
addpkg BetaSequences V00-06-01
addpkg workdir V00-02-04
```

It's a good idea to do the following when you check out workdir for the first time: cd to workdir and type “gmake setup”. You only need to do this once.

Which packages to add depends, obviously, on the nature of your work. For doing a Beta-type analysis you will need BetaUser and workdir at the very least. It is often helpful to check out related packages are well (e.g., BetaTools), so that they can be easily modified and used for expanding the functionality of your analysis.

6 A few words about Objectivity

6.0.2, as well as all the subsequent releases, requires Objectivity. Some information about Objectivity in BaBar computing is available at the following URL:

http://www.slac.stanford.edu/BFROOT/doc/Computing/Databases/www/frames.htm

There is also some documentation about it in our installation directory (on electron, go to /usr/object/alphaosf1/doc).

I will not go into any but the most basic details about Objectivity here. Basically, there are two things that you need to make sure are running when you do any

Objectivity-related work: the lock server must be up, and the AMS (Advanced Multi-Threaded Server) must be running. To check if the lock server is running, type (on electron):

```
/usr/object/alphaosf1/bin/oocheckls electron
```

which should tell you "The Lock Server is running (or is not running) on electron". If it's not running, you need to start it with the following command:

```
/usr/object/alphaosf1/bin/oolockserver
```

If the lock server was not running, chances are the AMS wasn't up either. You can start it by typing

```
/usr/object/alphaosf1/bin/oostartams
```

7 Compiling, linking, and running

I am assuming you are on electron in /babar2/you_directory/my6.0.2.

- Create a file named .bbobjy in your test release, and put your Federated Database (FD) ID there. Every BaBar user gets a set of 5 FD ID numbers; to find out what your numbers are, go to your SLAC home directory and type "GetFDID", which should tell you something like "The five FDID numbers assigned to your_user_name are 12345-12349". Let's suppose your first FD ID number is 12345. Then put the string "FD_NUMBER = 12345" into your .bbobjy file.

- Type "gmake database.config". If everything works correctly, you should get something like this:

```
Boot file (BOOT_FILE)   /usr/object/database/BaBar/your_user_name/BaBar
FDDB (FDB_DIR/FDB_NAME) /usr/object/database/BaBar/your_user_name/BaBar.FDDB
AMS Server (AMS_HOST)   electron.physics.ucsb.edu
Lock Server (LOCK_HOST) electron.physics.ucsb.edu
FDDB ID (FD_NUMBER)     1234
Page Size (PAGE_SIZE)   8192
Import Dir (IMPORT_DIR) /nfs/babar2/BFROOT/dist/releases/6.0.2/database/expor
Import Name             BaBarRef
Import FDDB             BaBarRef.FDDB
```

- Type “gmake database.import”, which should import the Federated Database (BaBar.FDDB), and load the conditions database. You can check whether this step worked correctly or not by going to /usr/object/database/BaBar/your_user_name. There should be the Federated Database BaBar.FDDB, and a conditions database directory, whcih in turn should contain drc and svt directories. Now, I get an error while loading the conditions database:

  ```
  ** Error #2174: ooHandle(ooDBObj)::open(): Object Manager: No Database
                  exists with name unkIndex.bdb
  Error: Can not open the unkIndex.bdb database
  load: Error during Conditions Database loading
  ```

  According to Stephen Gowdy, who is an OSF Objectivity expert, this error is unconsequential.

- Type “gmake clean”, which sets up clean lib, bin, and tmp directories.

- Now, there are three possible modes in which you can run the BaBar software: Alsund, which is the fast simulation, Reconstruction, where you generate and reconstruct your own data, and Beast, where you use tapes with events that have been generated and reconstucted for you. If you just used BaBarUCSB.cshrc to set up BaBar environmental variables, you are ready to go for Reconstruction and Beast jobs. I have made a few attempts to run Aslund in this release, but they were not successful. My impression is that nobody takes Aslund very seriously anymore, and I am not sure if anyone actually tried to make it work in this release. In what follows I will therefore describe how to run Reconstruction and Beast.

  Type “gmake CXX6=1 all”, which should compile and link everything that’s in your my6.0.2. This step might take a while. The new 6.0 DEC compiler generates a number of warnings at the compilation stage; they can be safely ignored. At the linking stage you should see “Linking with Reconstruction Beast for BetaUser”.

- In order to run Beast, you should first modify the beastSetup.tcl file in BetaSequences. Find the line that says “input file reco.rwend” and change it to “input file /nfs/data7/MDC1/GenericBBbar/reco046423.rwend”. The GenericBBbar directory on data7 contains 50 2,000 event beast tapes with $B^0 \to X$,

$\overline{B}^0 \to X$ events. I will explain how to get the tapes with the events you want later.

Now, cd to workdir, and type:

```
betarun ../BetaUser/beast.tcl
```

After getting a few initialization messages, you should see

```
beast.tcl script completed
>
```

At the > prompt, you can type "help" to see what options are available at this point. To simply run 5 events, type

```
ev begin -nev 5
```

and then after getting the > prompt again, when all 5 events have run, just type "exit".

- This trial run should produce an hbook file in workdir named (by default) framework.hbook. If you look at it in paw, "ldir" gives you something like this:

```
 ************** Directory ===> //LUN1 <===

                 Created 980430/1301  Modified 980430/1302


 ===> List of subdirectories
 EMCPIDHIST       Created 980430/1302 at record     4
 EMCIDENTIFY      Created 980430/1302 at record     6
 SAMPLEANALYSIS   Created 980430/1302 at record     8
 TRKQASVT         Created 980430/1302 at record    10
 TRKQADCH         Created 980430/1302 at record    12
 NEUTRALHADMATCH  Created 980430/1302 at record    14
 TRKQAMERGE       Created 980430/1302 at record    16
```

```
===> List of objects
   HBOOK-ID  VARIABLE                             CYCLE    DATE/TIME   NDAT
        1        0                                    1   980430/1302    64

 Number of records =    3  Number of megawords =  0 +  2688 words
 Per cent of directory quota used =   0.009
 Per cent of file used            =   0.009
 Blocking factor                  =  20.833
```

Here, "TRKQASVT", "SAMPLEANALYSIS", etc., are different subdirectories corresponding to different *modules*. I will talk about modules in more detail later. For now, do a cd to, say, SAMPLEANALYSIS. There will be a number of histograms there:

```
===> Directory :
        16 (1)   B mass
        18 (1)   di-neutral spectrum, mc match
        12 (1)   selected K0 momentum
         7 (1)   mass of all J/Psis
        13 (1)   selected K0 ctau
        10 (1)   correct J/Psi CM momentum
         4 (1)   mass of pairs
         9 (1)   all J/Psi CM momentum
        11 (1)   selected K0 pathlength
        14 (1)   B lab momentum
         3 (1)   No. track / event
         1 (1)   No. pion / event
         2 (1)   No. muon / event
        19 (1)   B x
         6 (1)   mass of Psi candidates
        20 (1)   B y
        15 (1)   B CM momentum
        21 (1)   B z
         5 (1)   mass of selected K0s
        17 (1)   di-neutral spectrum
         8 (1)   mass of correct J/Psis
```

Let us now see where these histograms were declared and how they were created.

8 Creating your own module

8.1 A brief overview of module structure

There is some documentation about the BaBar application framework on the Web at the following URL:

http://www.slac.stanford.edu/BFROOT/doc/Computing/NewUser/htmlbug/bug.html

and in particular Chapter 7 talks about modules.

You can also go directly to

/babar2/BFROOT/dist/releases/6.0.2/Framework/Framework

and then look at the corresponding header files, such as APPFramework.hh and APPModule.hh. This is tedious, but it might give you more detailed information about the module structure, as it is implemented in BaBar software. Finally, some useful module examples are located in

/babar2/BFROOT/dist/releases/6.0.2/FrameExample

Very briefly, the Framework package of BaBar allows one to handle basic event-processing tasks. The basic unit of the Framework is a *module*. All modules inherit from the base class APPModule in Framework, so, just like APPModule, they all have methods for initialization, event processing, and ending. The main user interface used by the Framework is the Tcl interpreter. It allows one to enable and disable modules, combine them into sequences and paths, set the number of events to process, etc..

Let us now consider an example. Go to /babar2/your_directory/my6.0.2/BetaUser. There is a great sample module there, SampleBetaAnalysis, written by Bob Jacobsen. Let's see what its main parts are. First, look at the header file, SampleBeta-Analysis.hh. You see in the class declaration (after the comment "class interface") that this class inherits from AppModule:

```
class SampleBetaAnalysis : public AppModule {
```

Then, there are five methods declared after the constructor and destructor:

```
virtual void           begin( AppJob* );
virtual void           begin( AppRun* );
virtual AbsEvent* event( AbsEvent* anEvent, int mode = 0 );
virtual void           end  ( AppRun* );
virtual void           end  ( AppJob* );
```

Here, "begin" is the initialization method, "event" is the main event processing method, and "end" is the termination method.

In the "protected" part of the program, there are a number of objects of the type "AbsParmIfdStrKey" (this class is also in Framework). These objects refer to various lists of BtaCandidates - e.g., _btaTrackList refers to a list of all charged tracks, _btaCalorList refers to a list of all calorimeter objects, etc. Finally, there are several pointers to the class HepHistogram, which (just as one would expect) allow one to create histograms by using the so-called *global* histogram manager declared in the .cc file. In section "Handling lists", we will add our own histogram to the existing ones.

8.2 A simple way to create your own module

Let us now try to create our own module. In my view, the simplest thing to do is to take an existing module and modify it. So, let's copy SampleBetaAnalysis.cc and SampleBetaAnalysis.hh to MySampleBetaAnalysis.cc and MySampleBetaAnalysis.hh, respectively. Now, let's first edit MySampleBetaAnalysis.hh, the header file. Let us make sure all the titles are right. Find the two lines at the beginning of the file which say:

```
#ifndef SAMPLEBETAANALYSIS_HH
#define SAMPLEBETAANALYSIS_HH
```

and change it to

```
#ifndef MYSAMPLEBETAANALYSIS_HH
#define MYSAMPLEBETAANALYSIS_HH
```

Now, after the comment "Class Interface", change the class declaration to

```
class MySampleBetaAnalysis : public AppModule
```

Similarly, change "SampleBetaAnalysis" to "MySampleBetaAnalysis" in the constructor and destructor, a little below.

Now let us change the .cc file. Find the comment that says "This Class's Header", and change it to

```
#include ``BetaUser/MySampleBetaAnalysis.hh''
```

Then, change "SampleBetaAnalysis" to "MySampleBetaAnalysis" in all instances (in the "constructors", "destructor", and the five methods or functions: two begin, event, and two end functions). Of course, everything that the module does is the same as what SampleBetaAnalysis does, but technically speaking we've just created a different module!

This module must be declared in AppUserBuild.cc in BetaUser. Edit this file, and add the line

```
#include ``BetaUser/MySampleBetaAnalysis.hh''
```

in the obvious place (e.g., immediately under a similar line for SampleBetaAnalysis). Then either comment out (with //) or delete the corresponding line for SampleBetaAnalysis (else paw later will get confused).

Then, a little below, where there are a number of such declarations, add

```
aModule = new MySampleBetaAnalysis( ``MySampleAnalysis'', ``my sample analysis'');
add( aModule );
```

and again, either comment out or delete the two similar lines for SampleBetaAnalysis.

Now, edit the file named "myAnalysis.tcl" (still in BetaUser), add the line

```
append MyAnalysis MySampleAnalysis
```

in the obvious place, and comment out or delete the line that says "append MyAnalysis SampleAnalysis".

At this point, you can compile (gmake BetaUser.lib) and link (gmake CXX6=1 BetaUser.bin) BetaUser again. If everything was done correctly, you should now be able to go to workdir, run Beast (betarun ../BetaUser/beast.tcl), and get a framework.hbook file, which should now contain a subdirectory named MYSAMPLEANALYSIS, like this:

```
PAW>


 ************** Directory ===> //LUN1 <===

                  Created 971128/1852  Modified 971128/1852


 ===> List of subdirectories
 TRKQASVT         Created 971128/1852 at record     4
 TRKQADCH         Created 971128/1852 at record     6
 TRKQAMERGE       Created 971128/1852 at record     8
 MYSAMPLEANALYSIS Created 971128/1852 at record    10
 BTAMCASSOCCHECK  Created 971128/1852 at record    15
 BTAPIDPLOTS      Created 971128/1852 at record    17
```

This subdirectory contains all the same histograms as we saw before with SAMPLEANALYSIS, of course.

To summarize, I think the easiest way to go about creating one's own module is to take a sample one (I have been using SampleBetaAnalysis for my work), modify it, and append it to the existing modules in AppUserBuild.cc and in the corresponding Tcl files.

8.3 Handling lists

Let us consider two very typical things that are used in almost any analysis. One is just getting a list of all charged tracks, and looping over it. The other one is getting Monte Carlo truth table information for the candidate particles, such as what they actually are and what their parents and daughters, if any, are. Both of these tasks are accomplished by using *lists* and *list iterators*.

- If you recall, in MySampleBetaAnalysis.hh we declared a list of charged tracks, _btaTrackList. Let us now loop over this list. To make things a little more interesting, let us at the same time create a simple histogram showing the momentum spectrum of all charged particles. To book the histogram, edit MySampleBetaAnalysis.hh and add the following line to it:

  ```
  HepHistogram*     _pspectrum;
  ```

(in the "protected" part of the code, where there are a number of such declarations.)

Now, in MySampleBetaAnalysis.cc, book this histogram like this:

```
_pspectrum = manager->histogram("p spectrum ",100, 0.0, 5.0);
```

after a number of similar lines. The arguments here are the histogram title, the number of bins, and the lower and upper limits.

In order to loop over all charged tracks, do the following. After getting eventInfo (the three lines after the comment "For convenience later, get a pointer to the event summary info object"), insert the following lines:

```
HepAList<BtaCandidate>* trackList;
getTmpAList (anEvent, trackList, _btaTrackList.value());
```

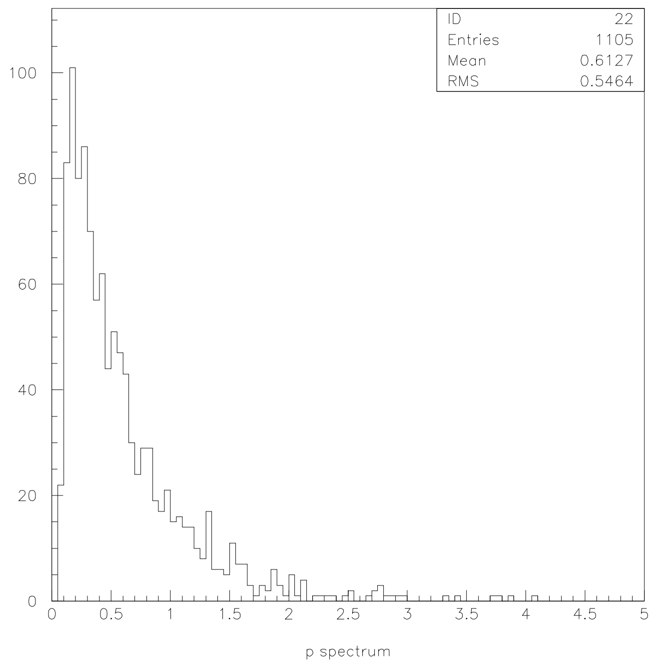
This defines a list of BtaCandidates which correspond to charged tracks. It is called trackList, and it is created from the (default) track list available via _btaTrackList. Now, to move around in this list and fill our momentum spectrum histogram, do the following:

```
HepAListIterator<BtaCandidate> iterList(*trackList);
BtaCandidate* trkPtr;

  while ( trkPtr = iterList() ) {

    _pspectrum->accumulate(trkPtr->p());

  }
```

(you can do this, for example, before the line "build a list of K0 candidates...").

This defines an *iterator*, iterList, which allows one to loop through the list. Lists and list iterators are described in the package called CLHEP (Class Library for High Energy Physics), so you can look at /babar2/BFROOT/dist/releases /6.0.2/CLHEP/Alist.

Now, after running Beast in the usual way (in workdir, betarun ../BetaUser/beast.tcl), we should get a histogram called "p spectrum" that should look something like this (for 100 events):



- To get Monte Carlo truth table information, we should first obtain a truth map, like this (these two lines are in MySampleBetaAnalysis.cc after the comment "get the default truth mapping") :

  ```
  BtaMcAssoc* truthMap = Ifd< BtaMcAssoc >::get( anEvent, "Default" );
  assert (truthMap != 0 ); // can't handle this gracefully yet
  ```

  It's a good idea to always check whether pointers are non-zero, by using assert or whatever method you prefer,- if they are, it will lead to a memory fault.

  Now we can do something like this (you can put these lines after the "p spectrum" while loop, for example):

```
HepConstAListIterator<BtaCandidate> iterList1(*trackList);
const BtaCandidate* trkPtr1;

     while ( trkPtr1 = iterList1() ) {

       const BtaCandidate* tmpMc;
       tmpMc = truthMap->mcFromReco(trkPtr1);

       //find out what this particle is
       if (tmpMc != 0) {
        cout << "This is particle " << tmpMc->pdtEntry()->lundId();
       } //end pointer non-zero loop


       //find out what its parent particle is
       if((tmpMc->motherIterator().next()) != 0) {
          cout << " which comes from " << (tmpMc->motherIterator().next()
                                          ->pdtEntry()->lundId()) << endl;


       //track list consists of stable particles, of course, but we can
       //find out what other daughters, if any, this parent particle has.

         cout << "This parent has number of daughters " << (tmpMc->
                      motherIterator().next()->nDaughters()) << endl;

      //loop over daughters to see what they are
      HepAListIterator<BtaCandidate> dauIter((tmpMc->motherIterator().next()-
                                                daughterIterator()));
      BtaCandidate* dauPtr;

          while(dauPtr = dauIter()) {
          cout << " daughters of this parent " << (dauPtr->pdtEntry()->
                                                 lundId()) << endl;
          } //end daughter while loop

       } //end parent pointer check loop
```

```
} //end the while loop
```

Every particle is assigned a unique PDT number, which is available at:

/babar2/BFROOT/dist/releases/6.0.2/PDT/PdtLund.hh,

and pdtEntry() -> lundId() in the code will give you these numbers, so that you can identify the particles. For example, a positive pion has lund ID number 211 (and pi minus, -211); electron has lund ID number 11; gamma, 22, and so on.

- As an aside, let us briefly consider what the following line in MySampleBeta-Analysis.cc means:

```
if(_verbose.value()) cout << name() << ": given " << trackList->length()
           << " charged candidates from " << _btaTrackList.value() << endl;
```

Here, name() refers to the name of the module (MySampleAnalysis in this case). If verbose mode for this module is set to true, this will produce lines like this for each event:

```
MySampleAnalysis: given 12 charged candidates from IfdStrKey(ChargedTracks)
```

You can set verbose to true in a suitable tcl script. For example, if you are running Beast, you can insert the following lines into beastSetup.tcl in BetaUser:

```
module talk MySampleAnalysis
       verbose set t
       exit
```

9 What are the Beast tapes?

There is a nice alternative to generating and reconstructing your own events. One can use the so-called Beast tapes, with events that have been both generated and reconstructed at SLAC during Mock Data Challenge 1 (MDC1). Some information about MDC1 is available on the Web, at the following URL:

http://www.slac.stanford.edu/BF/doc/Computing/Simulation/web/mdc1.html

You can browse the database to see which modes have been generated and reconstructed. If you would like to have your mode generated, contact the production managers, Adil Hasan (hasan@SLAC.Stanford.EDU) or Paul Raines (raines@slac.stanford.edu).

Assuming you are at SLAC, to see the status of a given run, e.g., 048010, type:

```
> dbreport 048010
```

which gives:

```
ORACLE_HOME set to /usr/oracle
  Run   Version    Tape  File  Nev  Description
=============================================================


048010  S4.3.7c   QJ1702   2   2k   B0~->D+ e- nueb  B0 ->X
        R4.3.8a   QJ1704   5
        R4.3.9a   QJ1065   7
```

To stage a run (again, at SLAC), type:
dbstage -r 4.3.9a 048010
This should produce a file with the extension .rwend, e.g., reco048010.rwend. Then you can transfer this file to UCSB (don't forget to use binary mode in ftp). These files are rather big: 5 of them (10k events total) take about 1GB. As I mentioned before, there are 50 tapes with generic $B\overline{B}$ events in /nfs/data7/MDC1/GenericBBbar.

10 Using EvtGen

EvtGen is an event generator that was created by Anders Ryd in association with a few other people. It implements most of the B and D decay modes that are of particular importance for BaBar in all detail. It is very convenient, easy to use, and (a rare thing for BaBar software!) has very clear documentation (which is in /babar2/BFROOT/dist/releases/6.0.2/EvtGen/doc or on the Web at http://www.slac.stanford.edu/BFROOT/doc/www/Physics/generators/EvtGen/EvtGen.html).

Let's suppose you would like to force $\Upsilon(4S)$ to always decay into B^0 and $\overline{B}^0$, with the B's decaying semileptonically, and the D's decaying into $K\pi\pi$. In order to do this, go to workdir, and create a file named UDECAY.DEC there. Here is what it might look like:

```
Decay Upsilon(4S)
1.000 2        B0   anti-B0                 VSS
Enddecay
Decay anti-B0
1.000 3 D+   e-   anti-nu_e                 ISGW2
Enddecay
Decay B0
1.000 3 D-   e+   nu_e                  ISGW2
Enddecay
Decay D+
1.000 3 K-   pi+    pi+                 D_DALITZ
Enddecay
Decay D-
1.000 3 K+   pi-    pi-                 D_DALITZ
End
```

The format for such files is the same as what is used in the generic decay table, DECAY.DEC, which is also located in the workdir, and the models (e.g., VSS, ISGW2, D_DALITZ) are described in the EvtGen documentation that I mentioned above.

Now, you also need to set an environmental variable, EVTUDECAY, equal to UDECAY.DEC, just like this:

```
setenv EVTUDECAY UDECAY.DEC
```

Now you are ready to generate your own events according to your decay table, UDECAY.DEC.

11 Running BBSIM

In order to run the full simulation, BBSIM, you have to first add a package called gnbase, so go to /babar2/your_directory/my6.0.2, and type "addpkg gnbase". Now just to be on the safe side, do a "gmake clean", after which you can do the following:

```
gmake include
gmake lib
gmake CXX6=1 BetaUser.bin
```

Now, assuming you want to generate your events with EvtGen, go to workdir and edit the file named sample.cards. Put the line “GEN 20” in it (e.g., after the line “C GEN 12 cosmic rays HemiCosm”). Also, find the line that says “TRIG 3”, and put whatever number of events you would like to generate instead of 3.

Now you can just type “bbrun” (all this in workdir), and this will generate your events. The output file is called (by default) bbsim.xdr. You can see the details of the run in bbrun.log. In particular, if you used a specific UDECAY.DEC file, you can look at the EvtGen messages at the very end of the file, and see whether the particles generated are consistent with what you put in UDECAY.DEC.

In order to reconstuct your events, type

```
betarun ../BetaUser/reco.tcl,
```

which should produce the usual framework.hbook file. This worked for me with the decay file, UDECAY.DEC, listed in the previous section.