Cluearte 0-42

The ARTE interface to BEE

HERA-B Note 01-110 / Software 01-015

Thorsten Glebe
Max-Planck-Institut für Kernphysik, Heidelberg
T.Glebe@mpi-hd.mpg.de

September 14, 2001

`cluearte` is an `ARTE` project dedicated to interface the `BEE` analysis library to the `ARTE` reconstruction framework.

Contents

1 Introduction

The `cluearte` package is part of the HERA-B reconstruction framework `ARTE` and has the task to provide an interface to the `ARTE`-independent analysis framework BEE [1]. BEE is an external shared library which is linked to ARTE at compile time. The location of the BEE library is at

```
/afs/desy.de/group/hera-b/BEE
```

As the cluearte code depends on the implementation of the BEE library, it is clear that a certain cluearte version can only support a limited set of BEE versions (as long as code changes within BEE do not affect the cluearte code). Therefore updating BEE can result in an update of the cluearte project within ARTE.

Recent cluearte versions can be found in the above mentioned BEE root directory. Usually a BEE version number is appended in order to indicate for which BEE library a certain cluearte version can be used, e.g. in the directory

```
/afs/desy.de/group/hera-b/BEE/cluearte_98
```

a cluearte version can be found which is suitable for BEE-0-98.

Note: When copying a cluearte directory to your local working directory, make sure that in your local cluearte directory a symbolic link

```
cluearte -> include
```

is set (if not, do it yourself via `ln -s include cluearte`). Otherwise compilation will fail because the cluearte header files will not be found. The reason for this is the structure of the ARTE include paths.

Note: The BEE and cluearte version which is at the moment (September 2001) supported by ARTE (all versions up to ARTE-03-09-r4) is outdated (the supported BEE release BEE-0-82 is ca. 18 month old and 98 revision steps behind the current release BEE-0-98[1]). However, this is not a limitation for the usage of BEE together with ARTE. Section 3 of this document describes the steps neccessary to compile ARTE with a recent BEE library.

Note: BEE is designed to work in ARTE as well as independent from ARTE in ROOT. In terms of speed it is advantegous for an analysis to use cluearte only to generate BEE files and then work locally in ROOT on these BEE files. Experience showed that this is much faster than running analysis code in ARTE.

2 ARTE, BEE and ROOT

Although BEE does not inherently depend on ROOT [2], BEE has an I/O layer which makes use of ROOT's TTree objects for an efficient and flexible data representation. Because of the BEE library it is neccessary to link ARTE with the ROOT libraries. The ROOT version which must be used for linkage depends on the ROOT version which was used to compile the BEE library.

[1] After BEE-0-82 the version number counting started for technical reasons again with 0-0.

The BEE-0-82 version which is used as a default in ARTE is compiled with ROOT 2.24.05. Newer BEE versions are compiled with newer ROOT versions. At the time this document is written (September 2001), the most recent BEE version (BEE-0-98) is compiled with ROOT 3.01.05.

3 Compiling ARTE with BEE

In order to make use of a BEE version which is more recent than the default BEE version supported by ARTE, some modifications to the user's ARTE-makefile are necessary. Eventually the ROOT library must be changed, too (see section 2). These modifications are described in the following. In order to find out which BEE version is supported by your ARTE version, look at the definition of the BEEVERS variable in the file `Makefile.setup`. This file can be found in the `makefiles` directory of the ARTE release used.

3.1 Changing the BEE library

In order to compile ARTE with a non-standard BEE library, several modifications in the user's Makefile are neccessary. In the following an ARTE Makefile from the ARTE `user` directory with the appropriate modifications is shown. In the example it is assumed that the default BEE version is replaced by the BEE-0-98 library.

```
############################################################
#
# This is a template local Makefile for compiling, linking
# and running your code within the HERA-B ARTE environment.
#
# Make sure the environment variables $HB (in your shell
# environment) and $HBVERS (set below) are set correctly.
#
# gmake # compile and link
# gmake -s # compile and link silently
# gmake run # compile and link (if necessary) and run
#
# More examples can be found in subdirectory "examples".
#
############################################################
#
# ARTE version:
HBVERS=ARTE-03-09-r3

# Name of the ARTE .kumac file:
kumac=mini
```

```
# Executable name:
exename=mcwrite

# Compiling and linking with/without the debugger option "-g":
debug=n
#debug=g

srcdirs = cluearte

include $(HB)/$(HBVERS)/makefiles/Makefile.setup

# Here is the place where experts can overwrite default settings
export HBDATAROOT=
srcdir=$(srcdir) $(HBROOT)/VDS/$(VDSVERS)/vdspatch

include $(HB)/$(HBVERS)/makefiles/Makefile.rules

###########################################################
# new BEE environment
###########################################################
beevers     = BEE-0-98
beepath     = $(HBROOT)/BEE/$(beevers)

beelibpath = $(beepath)/$(HBBINTYPE)/lib

beelibs  = $(addprefix -L,$(beelibpath))\
          -l$(beelib)

# All HERA-B libraries
hblibs     = $(artelibs) ${HBCLHEPLIB}/libCLHEP-$(cxx).a \
            $(vdslib) $(sltlib) $(datamgrlibs) $(databaselibs) \
            $(onlinelibs) $(beelibs) $(itrlib) $(otrlib) \
            $(rootlibs) $(cernlibs)

linkopt = $(hblibs) \
          -L/usr/X11R6/lib -lXmu -lXt -lX11 -ldl -lg2c -lcrypt \
          -lm -lXm -lnsl

LD_LIBRARY_PATH := $(beepath)/$(HBBINTYPE)/lib:$(LD_LIBRARY_PATH)


incdirs := cluearte $(HBROOT)/ARTE/$(HBVERS)/analysis/include \
           $(incdirs)
```

```
cxxopt := -ftemplate-depth-40 $(cxxopt)
```

3.2 Changing the ROOT library

In case an ARTE job needs to be linked with a ROOT version different from the default one (see section 2), only a few changes to the shell environment are neccessary. The default ROOT version which will be used by the ARTE Makefile is specified by the ROOTSYS environment variable, which is set in the hbprofile.sh script[2].

The cluearte project contains a small script which can do the appropriate changes to the shell environment which is called setroot and consists of a few statements like this:

```
ROOTSYS=/afs/desy.de/products/ROOT/3.01_05/gcc-2.95.2/root
LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
PATH=$ROOTSYS/bin:$PATH
```

The ROOT version to be used must be set by editing the setroot-file. To apply the changes to your current environment, just source the file:

```
bash> source ./setroot
```

After the shell environment has been changed, a remake of the ARTE job will result in a compilation with the new ROOT library.

4 Using BEE within ARTE

In order to make use of BEE from within ARTE, the user needs access to the BEE event structure. To generate a BEE event from an ARTE event the user has to call the clueGetEvent() function in his usevnt() routine.

The BEE event structure can be accessed in two ways:

1. The BEE ARTE tables. BEE represents a lot of information in the same way as ARTE does, namely via ARTE tables. These are global objects which can be accessed from anywhere in your code. For technical reasons, BEE-ARTE-Tables must be named differently as in ARTE. Example: To access reconstructed tracks, one can use in ARTE ArteTable<RTRA> and in BEE ArteTable<RecoTrack>.
2. The BEE event structure. The BEE event structure is represented by an object of type RtEvent (see the BEE manual). A pointer to the BEE event is returned by the clueGetEvent() function. Via the event pointer one has access to the event header information and to the track and vertex container without the ArteTable-interface. Example:

[2] See /afs/desy.de/group/hera-b/GLOBAL/pro/hbprofile.sh, or in case of tcsh the hbprofile.csh script.

```
// this is usevnt()
#include "BEE.hh"
...
RtEvent* event = clueGetEvent(...);
/// print the BEE Topp table:
cout << *event->topp() << endl;
```

It should be noted here that the `clueGetEvent()` routine, as well as the `clueWriteEvent()` routine, needs an object of type `TableFillBase` as an argument. How such an object can be defined will be explained in the next section.

Note: When calling `BEE` routines in your `usevnt()` routine, make sure that the first statement (execpt for comments) in `usevnt.C` is

```
#include "BEE.hh"
```

It is important that `BEE.hh` ist included before `arte.hh`, as `BEE` uses a slightly modified `arte.hh` (which can be found in the `arte` project of `BEE`).

After the call to `clueGetEvent()`, the user can make use of all `BEE` features within ARTE. In parallel he has full access to all `ARTE` tables. However, there is no direct connection between `ARTE` and `BEE` `ARTE`-Tables (that means no relations from `ARTE`-Table objects to `BEE`-Table objects do exist). A relation like `RTRA`↔`RecoTrack` can only be made in an indirect way.

5 Writing BEE files

The `cluearte` software allows the user to freely define event and track selection routines which are then used by `cluearte` to generate a `BEE` event from an `ARTE` event. The `BEE` event is stored in a `ROOT` file with a certain event structure (therefore called `BEE`-file) which allows later on an event-by-event analysis independent from `ARTE`. A `BEE` event has a dynamic event structure, that means the user has the possibility to store in addition to the `BEE` event structure an arbitrary amount of information per event. In order to write `BEE` events, the user has to call the `clueWriteEvent()` routine from his `usevnt()` routine. The `clueWriteEvent()` routine needs, as well as the `clueGetEvent()` routine (see section 4) an object of type `TableFillBase` as an argument.

The following sections will explain how the selection routines are defined and how additional user information can be stored.

5.1 The TableFill-Interface

The TableFill interface is implemented in the `TableFillBase` class, which can be found in the `TableFillBase.hh` file which is part of the `cluearte` project. The `TableFillBase` class defines three functions, which are called by the `cluearte` code:

`EventSelect()` In this routine the user can define event selection algorithms. The return type of this function is `bool`. In case `false` is returned by this function, the event will not be written to the `BEE` file.

`Fill()` This function will do the filling of the `BEE-ARTE`-tables. The `Topp` table is filled once in the beginning from the `ARTE` table `ArteTable<TOPP>`. The `BEE` event header information is filled automatically and cannot be steered by the user. The `Fill()` routine is called once per `clueWriteEvent()` call.

`FillRSEG()` In case the writing of RSEG information is enabled (see section 5.4), this routine will define the way how RSEG objects are stored as `BEE` segments. In contrast to `ARTE`, `BEE` has several classes to store segments: `VDS_Track`, `RICH_Track`, `OTR_SpacePoint` and `MuonTrack`. This function will be called once per `clueWriteEvent()` call.

5.2 User defined TableFill objects

The user can define his own event filling class by defining a new class which is inherited from the `TableFillBase` interface-class. As the functions defined by `TableFillBase` are pure virtual, the user must redefine them. In this way the `cluearte` software can call the redefined filling routines which are fully under user control.

A simple example is shown below:

```
#include "cluearte/TableFillBase.hh"
#include "BEE.hh"

//
// EventFill: a user defined event filling routine
//
class EventFill: public TableFillBase {
public:
  /// CTOR
  EventFill() {}

  /// Event selection: accept all events
  bool EventSelect() const { return true; }

  /// fill only RTRA table into BEE event
  bool Fill() const {
    const ArteTableIterator<RTRA> end = ArteTable<RTRA>::end();
    for( ArteTableIterator<RTRA> rtit = ArteTable<RTRA>::begin();
       rtit != end; ++rtit ) {
```

```
        const ArtePointer<RTRA> rtra = *rtit;

        ArteTable<RecoTrack>::push_back( RecoTrack(rtra) );
      }

      return true;
    } // end Fill()

    /// Do not fill RSEG's
    bool FillRSEG() const { return true; }

  }; // class EventFill
```

In `usevnt()`, an object of type `EventFill` must be created (as a static member, in order to avoid useless instantiations) and then passed to `clueWriteEvent()` or `clueGetEvent()`:

```
  // this is usevnt.C
  #include "EventFill.hh"

  int usevnt() {
    ...
    /// user defined fill object
    static const EventFill MyFillObj;
    /// pass user defined filling routines to cluearte
    clueWriteEvent(MyFillObj);
    ...
  }
```

In `cluearte` two event filling classes are already provided. They can serve as an example for more complex filling routines:

`TableFill` The `TableFill` class is the most simple but complete way of generating a `BEE` event from an `ARTE` event. For all tables which are supported in `BEE`, a one-to-one copy is made from the correponding `ARTE` table.

In case of MC an additional feature is supported: The user can specify via a `TableFill` constructor call an energy threshold for MC tracks. Then only those MTRA tracks are stored in `BEE` (as `MC_Track` objects) which have an energy above the specified threshold. This is useful if one wants to avoid that a lot of low-energetic tracks from shower simulations are stored. Experience shows that an energy threshold of 0.1 GeV reduces the size of MC files by a factor of 2. In case MVER objects become obsolete (because all their outgoing MTRA tracks are skipped due to the energy threshold) they are not stored in `BEE` (as `MC_Vertex` objects). Example:

```
// this is usevnt.C
#include "cluearte/TableFill.hh"

int usevnt() {
  ...
  // set MC energy threshold to 100 MeV
  static TableFill FillObj(0.1);
  clueWriteEvent(FillObj);
  ...
}
```

`SlTableFill` The `SlTableFill` class is developed for writing `BEE` files for the Single Lepton analysis. In the `EventSelect()` routine the `anl_evselect()` routine from the `ARTE analysis` project is used. In case of RTRA's, the `Cloneremove()` routine [3] is called and VDS only, VDS-ECAL and VDS-ECAL-RICH matches are ignored. A bookkeeping of clones in the `BEE TrackClones` table is done (which has no correspondance in `ARTE`). Only primary vertices are stored in the `BEE RecoVertex` table. The primary vertex search is redone by Grover by only considering tracks which have passed the clone removal scheme.

5.3 Storing user data in a BEE event

As already mentioned, the `BEE` event structure is **dynamic**, that means the user has the possibility to store an arbitrary amount of additional information in the `user` field of the event header. This `user` field is implemented as a `STL` [4] `vector<float>` object and is thus free-format. The user has to decode the informations (and the order of informations stored) by himself. The amount of information stored can vary from event to event.

In order to write data to the `user` field, the user has to define a class for writing `BEE` events (see section 5.2). To get access to the global event header pointer, an `extern` statement must be used. Then the user field can be used as an ordinary `STL` vector of type `float`:

```
#include "cluearte/TableFillBase.hh"
#include "BEE.hh"

// pointer to BEE event header
extern RtEvent* clueEvent;

//
// EventFill: a user defined event filling routine
//
class EventFill: public TableFillBase {
```

```
public:
  /// CTOR
  EventFill() {}

  /// Event selection: accept all events
  bool EventSelect() const { return true; }


  bool Fill() const {
    /// store length of certain ARTE tables in user field
    clueEvent->user.push_back(ArteTable<RTRA>::size());
    clueEvent->user.push_back(ArteTable<RVER>::size());


    /// fill only RTRA table into BEE event
    const ArteTableIterator<RTRA> end = ArteTable<RTRA>::end();
    for( ArteTableIterator<RTRA> rtit = ArteTable<RTRA>::begin();
       rtit != end; ++rtit ) {
       const ArtePointer<RTRA> rtra = *rtit;

       ArteTable<RecoTrack>::push_back( RecoTrack(rtra) );
    }

    return true;
  } // end Fill()

  /// Do not fill RSEG's
  bool FillRSEG() const { return true; }

}; // class EventFill
```

5.4 ARTE kumac settings

Several `kumac` parameters are available to steer the `cluearte` behaviour:

`CLUE/SETFILE` This card sets the name of the `ROOT` file which will be written by `cluearte`.

`CLUE/StoreRSEGflag [1|0]` Write RSEG contents into ROOT file, if SWITCH=1.

`CLUE/SetBranchStatus [bname]` Set branch status `Process` or `DoNotProcess`. `bname` is the name of a branch. If `bname="*"`, apply to all branches.

`CLUE/SetCompressionLevel [1...9]` Sets the compression level for the `ROOT` file (values belong to the `gzip` compression levels).

References

[1] http://www.mpi-hd.mpg.de/herab/clue

[2] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.

[3] M.-A. Pleier, Cloneremove V1.0, HERA-B Software Note 01-062.

[4] Stepanov, A. and Meng Lee, The Standard Template Library, ANSI X3J16-94-0095 / ISO WG21-NO482