CLAS NOTE 98-016
CVS: docs/clas_notes/clasbanks

Bank Classes for CLAS Banks

Mark M. Ito

October 16, 1998

1 Introduction

I have written a perl script that will produce C++ classes that can be used to access data in our BOS banks. Here is an example of a C++ function that uses one of these classes to print the event number:

```
#include <iostream.h>
#include "clasbanks.h"

void print_event_no(void) {
  int my_event; // Local copy of the event number
  HEAD my_head(0); // Declare a local instance of the "HEAD" class. The 0 is
                   // the BOS bank index.
  my_event = my_head.get_NEVENT(0); // Get the event number. The 0 is the row
                                    // index of the BOS bank.
  cout << "The event number is " << my_event << "\n"; // print the number
  return;
}
```

The perl script is run automatically by the complete library build and produces both the header file and the C++ code needed to construct the classes, starting from the data definition language (ddl) files in the packages/bankdefs directory. This type of thing has already been done by Arne F. for FORTRAN and by Joe M. for C.

2 Details of Use

2.1 Coding

Since the code is generated from the ddl files, you can consult the ddl files themselves to figure out the variable and function names. Continuing with the HEAD example, the relevant ddl file is packages/bankdefs/head.ddl:

```
!-----------------------------------------------------------------------
!       BANKname BANKtype      !Comments
 TABLE   HEAD      ! create write display delete ! Bank HEAD
```

```
!
!COL ATT-name FMT Min    Max   !Comments
   1 VERSION  I    0      2    ! Version Number
   2 NRUN     I    1      1000 ! Run Number
   3 NEVENT   I    1      1000 ! Event Number
   4 TIME     I  100 100000000 ! Event Time (UNIX time)
   5 TYPE     I    0      1000 ! Event Type (Defined by on-line system or MC run)
   6 ROC      I    0      20   ! 32 bit readout controller status
   7 EVTCLASS I    1      20   ! Event Classification from DAQ:
   8 TRIGBITS I    0      4096 ! Trigger Latch Bits
                               !
!
 END TABLE
!
```

The "TABLE" line tells you that the class name is HEAD. There is a "get" member function for each column of the bank, in our example get_NEVENT for the event number. There is no fooling with capitalization; names appear exactly as in the ddl file.

The integer argument in the constructor is the bank index and is passed directly to the BOS routine which locates the banks in the bcs common. "0" works for banks that only have one instance. If the bank is an one-per-sector type (DC0 say), the appropriate argument is one of $1, ..., 6$.

The integer argument in the get function is the row index of the bank, counting from 0. The total number of rows in the bank is available via a get function, for example:

```
int nhits;
DC0 my_dc0(4); // gets sector 4
nhits = my_dc0.get_nrows(); // # of rows in DC0 = # of hits in sector
```

2.2 Linking

The compiled code for the constructors are in an object library, $CLAS_LIB/libbankdefs.a. The header file is in packages/inc_derived/clasbanks.h. Both of these are made automatically by the complete build.

3 Examples of Generated Code

The code needed to declare the HEAD bank class in clasbanks.h looks like:

```
struct HEAD_row {
  int VERSION;
  int NRUN;
  int NEVENT;
  int TIME;
  int TYPE;
  int ROC;
  int EVTCLASS;
```

```
  int TRIGBITS;
};

class HEAD {
private:
  int nrows;
  HEAD_row *row_ptr;
public:
  HEAD(int bank_index);
  inline int get_nrows(void);
  inline int get_VERSION(int row);
  inline int get_NRUN(int row);
  inline int get_NEVENT(int row);
  inline int get_TIME(int row);
  inline int get_TYPE(int row);
  inline int get_ROC(int row);
  inline int get_EVTCLASS(int row);
  inline int get_TRIGBITS(int row);
};

inline int HEAD::get_nrows(void) {
  return (nrows);
}

inline int HEAD::get_VERSION(int row) {
  return ((row_ptr+row)->VERSION);
}

inline int HEAD::get_NRUN(int row) {
  return ((row_ptr+row)->NRUN);
}
```

and so on for all of the rest of the get functions.

The HEAD section of the constructor code looks like:

```
HEAD::HEAD(int bank_index) {
  int bos_ind, ind;
  bos_ind = get_bos_index("HEAD", bank_index);
  if (bos_ind) {
    ind = bos_ind - 1;
    nrows = bcs_.iw[ind - 4];
    row_ptr = (HEAD_row *)(&bcs_.iw[ind + 1]);
  } else {
    nrows = 0;
    row_ptr = NULL;
  }
}
```

Note that “get_bos_index” is a C++ function.

4 Code Generation Process

During the complete build, a "make lib" is done in the bankdefs directory. Among other things, this invokes the perl script packages/scripts/clasbanks.perl. The script looks through all local ddl files and creates two files: clasbanks.h and clasbanks.cc. The former contains the class definitions and the inline get functions, the latter the constructor code. clasbanks.h is then moved to packages/inc_derived. clasbanks.cc, get_bos_index.cc and make_bos_call.F are then compiled and put in $CLAS_LIB/libbankdefs.a.

5 Current Limitations

As of this writing there are some significant limitations.

- The classes work on the iw array only.
- Character data in the banks is not handled properly.
- Non-full-word data is not handled properly.

If you try any of these things, you will get unpredicable results. Since the stucture overlay occurs on a bank-by-bank basis, if the bank of interest is not a problem, you should not have a problem.