CLAS Event Format with BOS

Version 1.00 (October 1, 1994)

Dieter Cords[1], Larry Dennis[3], Dave Heddle[2], Bogdan Niczyporuk[1]
[1] CEBAF, [2] Christopher Newport University, [3]Florida State University

1.0 Introduction

For a detector of the complexity of CLAS, which is expected to operate over many years, one needs a scheme for dynamically managing data structures. These data structures, in the end, will contain information on raw events, detector status, geometry and calibration constants, reconstruction results for tracks and clusters, as well as analysis results the user chooses to append. This note describes the structure of raw events, i.e. a naming convention for major detector components and a numbering scheme which reflects the arrangement of the smallest detector elements.

2.0 Numbering Scheme

The numbering scheme follows the convention adopted by the CLAS collaboration some time ago. Based on a right-handed coordinate system with Z along the beam and Y up, the detector elements are numbered in the order of increasing R, polar angle Θ, and azimuth angle Φ (ranging from 0 to 2π). As usual, Θ counts from the Z-axis and Φ from the X-axis (in a clockwise sense when viewed along the Z-axis)

.

3.0 Management of Data Structures

The package for managing data structures for CLAS is BOS with FPACK as its I/O front end. The basic unit of information is contained in a ***bank*** which is identified by a name (up to 4 characters) and optionally a number. For details the reader is referred to the manuals in CLAS_doc (one of the symbolic Unix links defining the CLAS environment). The module libraries differ for various machine architectures and can be found in CLAS_lib which points to a platform-dependent section of the Unix directory.

It is anticipated that application modules - ranging in seize from a small set of routines (e.g. for histogramming detector hits) to complex packages (for track or cluster reconstruction) - will receive all experiment-specific information in the form of BOS banks.

The communication between modules will be done exclusively by BOS banks; and if an application produces information for any CLAS data stream or data set, it has to be in the form of BOS banks. In order to achieve and maintain a modular structure of the CLAS software, one should strictly avoid to exchange information between application modules by passing parameters in function or subroutine calls or by using special common blocks. Never should any experiment-specific information be buried in Fortran data statements.

The CREATION of banks is assigned to specifically designated program modules and other modules should ***never*** change a bank received for input. The raw data banks are created by the on-line "Event Builder", or in case of detector simulation by the originating Monte Carlo program.

4.0 Types of Raw Event Banks

It is customary to start the array of raw data banks with a header bank (name: HEAD, number: 0). The contents of the header bank will be discussed in section 8.1 where the control and status information is reviewed which accompanies individual events.

For CLAS the various detector components are associated with the following bank names and numbers. Further banks are added as needed.

Detector Components	**NAME**	**NR.** (sectors)
Drift Chamber	DC	1....6
Cerenkov Counter (forward)	CC	1....6
Cerenkov Counter (large angle)	CC1	1....6
Scintillation Counter (time-of-flight)	SC	1....6
Electromagnetic Calorimeter (forward)	EC	1....6
Electrom. Calorimeter (large angle)	EC1	1....6
TriGger	TG	0
TAgger	TA	0

An event is then given by a set (ensemble) of banks with at least one data item (hit) each. Empty banks are omitted.

For Input/Output to external media the banks are packed in data segments and the set of data segments, which form an "event", are preceded by a logical record header (see FPACK manual). This logical record header contains - as part of a search key - the run type (up to 8 characters), e.g. 'RUNEVENT' or 'MCEVENT', as well as the run number and event number.

5.0 Contents of Raw Event Banks

The experimental data are stored inside the banks in tabular form, where the number of columns NCOL is determined by the number of parameters, which characterize a detector component, and successive rows are filled with successive hits in that particular detector volume. The size of the table (NCOL,NROW) is kept in the data segment header of that bank. All raw event data are packed/stored as 16-bit integers.

Here we list the above bank names again, followed by the parameters for the detector elements. NCOL is either 3 or 5 for the banks considered. The first column always contains the address ID of the hit detector element.

DC: ID TDC ADC

with ID = 256*L + W & L = 1...36 (layer #), W = 1...192 (wire #)

Note: Superlayers are made up of 6 Layers (however L = 5 & 6 do not exist). Wires with the same number are intended to line up radially in the same superlayer. To achieve this, the numbering starts with W=4 for L=1&7, W=3 for L=2,3,8,9 and W=2 for L=4,10,11. All other layers start with W=1.

CC: ID TDC ADC

with ID = 1...36 & Odd/Even = Left/Right (Low Φ / High Φ seen along beam)

CC1: same format as for CC

SC: ID TDC_L ADC_L TDC_R ADC_R

with ID = 1....48

EC: ID TDC ADC

with ID = 256*L + S & L = 1....6 (ganged layers u,v,w,u,v,w)
S = 1....36 (strip #)

EC1: ID TDC_L ADC_L TDC_R ADC_R

with ID = 256*L + S & L = 1....4 (ganged layers x,y,x,y)
S = 1....40 (x strips for L odd)
S = 1... 24 (y strips for L even)

TG: *** format to be decided later ***

TA: *** format to be decided later ***

6.0 Bank Format and Table Dimensions

The raw event data - as described in the previous section - are always packed in 16-bit words, or in the parlance of BOS have the format B16. It is anticipated that all reconstruction and analysis results will be stored in 32-bit words, either integer or floating point - and, of course, one can mix integer and floating point in one bank. However, it is important that one declares - by a call to BKFMT(Name,Fmt) - the format of any newly created bank if one wants to store it on disk or tape. FPACK **needs** this information to make the I/O truly platform-independent.

All experimental data will be kept in tabular form. We already discussed this for the raw data in the previous section. The important parameter for a bank is the number of columns, NCol, which determines the number of parameters per hit, track, cluster and so on. The number of rows - which can be derived from NCol and the total number of data words, ND, in a bank - simply gives the number of hits, tracks or clusters. One can declare NCol for a bank by a call to BKCOL(Name,NCol) [routine to be written]. If one wants to find out the number of columns in a bank, one can use the function NCol = NBCOL(Name) [function to be written].

When one declares the format (integer and/or floating point) of a bank, one *only* has to do it for NCol items in the first row. BOS/FPACK implies that this format is repeated indefinitely for all the hits, tracks etc. in a bank. One should also keep in mind that all banks with the same name (but different numbers) can only have one format and always have the same number of columns. From an organizational point of view it is, of course, simplest if one avoids mixed formats and declares a bank to be either integer or floating point. Then one only needs a single character - I or F - to declare the format of a whole bank.

7.0 Linking Libraries and a Simple Examples

To build a program with BOS/FPACK calls, one needs the libraries libh1util.a, libbos.a and libfpack.a in CLAS_lib (one of the symbolic Unix links defining the CLAS environment). There are several high-level routines which presently reside in libh1util and combine a number of functional calls to libbos and libfpack. These can be used to custom-tailor the management of data structures for CLAS. For instance, one may want to customize the print-out of the collection of banks in an event.

In the following two sections we shall discuss how to use BOS/FPACK within Fortran and “C” programs. The Fortran example is shorter because BOS/FPACK was designed for a Fortran environment. But the “C” example is quite instructive and shows how convenient C structures can be.

7.1 Using BOS and FPACK from Fortran Programs

In this example we use the high-level routine FSEQR - for sequential reading of events - to illustrate the basic use of BOS:

```
      PARAMETR  NBCS=500000
      COMMON /BCS/ IW(NBCS)
      EQUIVALENCE (IW(1),RW(1))

      CALL BOS(IW,NBCS)                                           ! initialize BOS
      CALL FPARM('OPEN BOSINPUT  FILE="..." READ ....  ')         ! open for input

10    CALL FSEQR('BOSINPUT',IRET)        ! read event
      IF (IRET.lt.0) GOTO 100

        IND  = NLINK('MCIN',0)      ! find MC input track bank
        ND   = IW(IND)              ! get length of data array inside of bank
        FIRST = RW(IND+1)           ! access first word in data array as floating point

       .......

      GOTO 10

100   CALL FPARM('CLOSE')       ! close files
```

7.2 Calling BOS and FPACK from “C” Programs

It is possible to make direct calls from “C” to the BOS/FPACK libraries. By “direct” calling we mean that no additional FORTRAN wrapper is required.

There are six rules to keep in mind:

1) All FORTRAN names, including subroutine, function and COMMON block names are converted by a preprocessor to lower case. Thus regardless of how they appear in the BOS and FPACK user manuals, the “C” code must access them via exclusively lower case names.

2) All FORTRAN names, including subroutine, function and COMMON block names receive an appended underscore character “_” . The “C” code must explicitly attach the underscore.

3) FORTRAN expects all arguments to be passed by reference, thus the “C” calls must use pointers.

4) Character strings must be handled with extreme care. For every character string

argument in the FORTRAN subroutine the “C” implementation must add an additional argument: the length of the string. The lengths which are passed by value, always appear at the end of the argument list in the same order as the strings themselves. For example

SUBROUTINE MySub(STR1, ARG2, ARG3, STR2) in FORTRAN is called via:

mysub_(str1, &arg2, &arg3, str2, strlen(str1), strlen(str2)) in “C”.

5) COMMON blocks in FORTRAN become global structures in “C”.

6) (Not applicable for BOS/FPACK but included for completeness) The storage of arrays is column-major in FORTRAN and row-major in “C”. Thus the indices must be reversed.

In the next subsection we provide some examples of these rules.

7.2.1 Sample of “C” code calling BOS and FPACK subroutines

7.2.1.1 BOS Banks and COMMON Blocks

Here is how the main BOS bank, which is stored in a COMMON block named BCS, can be implemented in “C”.

a) First define a BOSBank data type:

```
#define  NDIM 20000
typedef struct bosbank {
  int    iw[NDIM];
} BOSBank;
```

then declare a global variable:

```
BOSBank  bcs_;   /* corresponds to COMMON BCS */
```

Data can then be accessed directly, e.g. bcs_.iw[9] gives the tenth word stored in the COMMON block. The size of a bank whose index is *ind* is in bcs_.iw[*ind* - 1].

7.2.1.2 Three more examples

```
char   *dataname = “BOSINPUT.DAT”;
char   *command;
int     iret, ind;
int     j = 3;
```

```
fseqr_(dataname, &iret, strlen(dataname);  /* calls FPACK FESQR */

ind  = nlink_("DC  ",  &j, 4);  /* calls BOS NLINK to get index to j'th named bank */

fparm_(command, strlen(command));   /*call the FPACK interpreter FPARM*/
```

7.2.1.3 Effective use of pointer casting for easy data access.

The data in the banks can be accessed elegantly by declaring a structure that matches the definitions of the banks. However, some knowledge of the byte ordering is necessary. For example, the declaration

```
typedef struct dcdata  *DCDataPtr;

typedef struct dcdata
{
  unsigned char  layer;
  unsigned char  wire;
  unsigned short tdc;
  unsigned short adc;
} DCData;
```

allows us to do the following:

```
DCDataPtr      dchits;
int            ind;
int            i, j;
int            banksize;
int            numhits;

ind  = nlink_("DC  ",  &j, 4);               /*  index to jth bank */
banksize = bcs_.iw[ind-1];                   /*get the bank size*/
numhits = banksize/sizeof(DCData);   /*see how many hits*/
dchits = (DCDataPtr)bcs_.iw[ind];      /* cast  first data word*/
```

then the data for the i'th hit can be accessed as:

```
dchits[i].layer
dchits[i].wire
dchits[i].tdc
dchits[i].adc
```

7.2.2 Sample *Makefile* for linking BOS and FPACK to a "C" program

Listed below is a *Makefile* for creating the application *ber*, a bOS eVENT rEADER.

ber consists of the three files: *bos_main.c, bos_utilities.c, and bos_event.c.* The *Makefile* shows how the BOS/FPACK libraries *libh1util.a, libbos.a, libfpack.a*, as well as the CERN library *libpacklib.a* are linked.

This *Makefile* has been tested only on the HP system. Potential machine dependencies are:

1) The library *libcl.a* is an HP-specific library required by "C" programs that make FORTRAN calls.

2) On some systems, it may be necessary to use f77 rather than cc to link all the object modules into an executable.

```
#---------------------------------------------------------------------
#define the C compiler

    CC = cc
#---------------------------------------------------------------------
# This variable contains the flags passed to cc

    CFLAGS = -O -c
#---------------------------------------------------------------------
# This variable points to the main library directory

    LIBDIR = /usr/lib

#---------------------------------------------------------------------
# This variable points to the main include directory

    INCLUDEDIR = /usr/include

#---------------------------------------------------------------------
# This variable lists the standard C libraries that must be loaded. (note that libcl.a
  is required for the fortran-C interface)

    STDLIBS =  -lcl -lm

#---------------------------------------------------------------------
# These variables list bos path and libraries

   BOSPATH = /usr/site4/classw/bosfp/lib.hp [to be changed with new directory structure]
   BOSLIBS =  -L$(BOSPATH) -lh1util -lbos -lfpack

#---------------------------------------------------------------------
# These variables list cern path and libraries.

   CERNPATH = /usr/site3/cern/hp700_ux90/94a/lib
   CERNLIBS = -L$(CERNPATH) -lpacklib
```

```
#-------------------------------------------------------------------
# This variable lists ALL libraries that must be loaded.

    LIBS = $(BOSLIBS) $(CERNLIBS) $(STDLIBS)

#---------------------------------------------------
#program  name

    PROGRAM = ber

#-------------------------------------------------------------------
#the source files that make up the application

    SRCS = bos_main.c bos_utilities.c bos_event.c

#-------------------------------------------------------------------
# The object files (via macro substitution)

    OBJS = ${SRCS:.c=.o}

#-------------------------------------------------------------------
#how to make a .o file from a .c file

.c.o :
        $(CC) $(INCLUDES) $(CFLAGS) $<

#-------------------------------------------------------------------
# This rule generates the (optimized) executable using the object files and libraries.

    $(PROGRAM): $(OBJS)
        cc -o $@ $(OBJS) $(LIBS)

#-------------------------------------------------------------------
#additional dependencies

bos_main.o:       bos.h
bos_utilities.o:  bos.h
bos_event.o:      bos.h
```

8.0 Definition of Related Banks

An important feature of BOS/FPACK is to access events or other data selectively. On an external medium (disk or tape) the events carry a logical records header, followed by a set (group) of data segments (see FPACK manual). With the data segment headers stripped off and converted to bank headers, this set (list) of banks is presented to the user in the Common BCS as an event or other collection of data elements. The selection process can occur before the event is read in completely by using information in the logical record header. This information consists of the run type, run number, event number and event classification. The run type is a string of up to 8 characters, e.g. 'RUNEVENT', 'RUNCALIB', 'RUNTEST', 'MCEVENT' etc.. Run and event numbers are integers and the event classification is a bit string (up to 31) which will be defined later and may contain some hardware and software "trigger" bits.

Some of the information in the logical record header is available to the user in the Event Header Bank which by convention is the first bank to be packed into or extracted from a logical record. This HEAD bank contains additional information which is of central importance for the event and which can be used to further classify or select the events.

In the following, the contents of the HEAD bank will be presented as well as the contents of the MCIN bank which describes the kinematic input to Monte Carlo programs which in turn produce "simulated" raw event banks.

8.1 Contents of the Event Header Bank

This section describes the format of the event headers to be used with CLAS events written in the BOS format. The major considerations involved in setting this format are providing for the begin-run, end-run and control data processing desired and being able to use some of the functionality for automatic event sequencing provided by the FSEQx routines (x = R or W for Read or Write).

BOS treats records as the basic building blocks for I/O on secondary storage, CODA I/O operates over network links where it uses event streams which contain event banks. In our event processing applications we deal with logical records, which contain a collection or set of banks, which we refer to as "events".

The FSEQx routines place some restrictions on the contents of event headers, namely, they must contain the Run number and the event number. CODA puts several different types of events in the event stream. These include physics events, prestart events, go events, pause events and end events. Which of these events ends up on the output stream is user configurable and will be decided by the on-line standard operating procedures.

To accommodate all of these event types, the BOS bank contents shown in the table on the **next page** will be used. This bank is named 'HEAD', and by convention is the first bank in a logical record.

HEAD Bank

Column (word)	Parameter	Format	Description
1	Version Number	I	Version of event format
2	Run Number	I	Monotonically increasing
3	Event Number	I	Starting with 1 at run begin
4	Event Time	I	Unix time = seconds as of January 1, 1970
5	Event Type	I	Defined by on-line system or MC run: =0 Control Records >0 Triggered (Physics) Events <0 Monte Carlo Events: -1 = SDA -2 = GEANT -3 = ClasSim
6	ROC Status	I	32 bit readout controller status*
7	Event Class	I	Event Classification from event builder containing the 4 bit trigger code plus user defined bits.* This translates into physics events from 1 to 15 and CODA control flow records with values greater than 15. 1-15 Physics Events 16 Sync Event 17 Prestart Event 18 Go Event 19 Pause Event 20 End Event
8	Event Weight	I	Prescale factor for this Event Class (Trigger Type) = Number of triggers to get an event.

* See the Appendices in the CODA manual for details.

The HEAD bank consists only of Integers and is followed by other banks. The Version Number will initially be set to zero. Experimentally triggered and simulated events supply the raw event banks discussed above. In the experimental setting one will have a number of “control” events interspersed with “physics” events (raw data). Some of these control events may arise from the CODA control sequence and are identified as shown in the table above.

8.2 Contents of Monte Carlo Input Bank

To evaluate the combined effect of detector emulation and event reconstruction, the Monte Carlo procedures should supply the kinematic track parameters obtained from the event generator. This information has to be stored in the 'MCIN' bank which has the following structure:

MCIN: X_1 X_2 X_3 C_1 C_2 C_3 P M Q LID

with X_i = event coordinate at target location (x, y, z in section 2)
C_i = direction cosines
P = momentum
M = mass
Q = charge
LID = Lund particle ID

For n generated particles there are n such rows in the MCIN bank. The format of the bank has to be declared as '9F,I'. The particle ID for the electron for example can be obtained by the function call LID = JPCODE('e-'). This function is an Entry to the PMASS subroutine in libh1util.a. Other standard particle character strings are 'e+', 'pi-', 'pi+', 'p+', 'n', 'gamma' etc.