The BOS System

Dynamic memory management

Third updated printing

Volker Blobel
II. Institut für Experimentalphysik
Universität Hamburg
Luruper Chaussee 149
2000 Hamburg 50

January 20, 1994

Abstract

BOS is a program system, written in standard FORTRAN77, for the dynamic management of data areas and for the input/output of sets of data areas. The system supports a modular structure of the application program and portability for both the software and the data sets. The main application is in the data analysis of high-energy-physics experiments.

Contents

List of Tables

1 Dynamic memory management

In the world of scientific computing the most popular programming language is FORTRAN. It is easy to learn, produces efficient code for numerical computations and, last but not least, it is the traditional programming language in science. It has however, like other programming languages, certain limitations for the analysis of large and complicated data samples, that are generated in big scientific experiments. The basic limitations in this respect are:

- fixed dimensions of data arrays, and
- fixed list (of arrays) in input/output.

The fixed dimensions of array are not well suited for the many different variable-length data areas in certain applications. The fixed list of arrays in input/output requires in principle, that the program in advance has information about the content of the next record to be read and processed.

These limitations are even more severe, when the whole analysis software, perhaps consisting of as many as 100000 lines of code, is in a continuous development. Another problem, beyond the programming language, is the portability of data sets, which is complicated by the variability of the machine representation of data on different types of computers.

An obvious way to improve the situation is the organisation of the different data areas dynamically by a system, which can be realized as a set of FORTRAN subprograms. In these dynamic memory management systems the data areas are usually called banks; each bank contains data, which belong close together. All banks are stored in one large COMMON area. The basic operations, provided by the system, are:

- create a bank,
- find a bank,
- delete (drop) a bank, and
- garbage collection.

In the garbage collection the existing data banks are compressed allowing new data banks to be created. The main advantage of dynamic memory management systems is the fact, that they allow a certain structure for all data, which is an esential condition for a modular structure of analysis programs.

The BOS system for the dynamic memory management was originally written in 1975 for the use by the PLUTO collaboration at DESY, but has been used afterwards by several high-energy-physics experiments in various countries. It includes in its design the input/output of data banks for sequential and direct-access data sets. This article describes the FORTRAN77 version, which is based on the same principles as the first version, but has been completely reorganised and improved with respect to portability of both program code and data sets. It is compatible in the respect, that sequential data sets written with the old version of the BOS system can be read. The main new features are the following:

1. More than one array can be used.
2. An additional type of banks, the work bank, is introduced, especially for cases, where the number of data words in a bank is frequently changed.
3. The system is written in standard FORTRAN77 (ANS FORTRAN 1978 Standard), a few machine dependent library functions are used.
4. A format can be specified for banks (necessary for machine independent I/O).
5. Machine independent I/O is introduced.

In the FORTRAN77 version of the BOS system up to 10 arrays, called BOS arrays JW, can be used to store banks, one of which has to be the basic array IW in the COMMON BCS:

```
      COMMON/BCS/IW(ndim)               ndim = constant
```

In most applications it will be sufficient to use only this basic array, which is used by the system for all input/output buffers. There are banks of two types, named banks and work banks, stored in each array in the low index part and the high index part, respectively. The structure of the arrays is shown below.

index	content
1 ... 50	systems data (50 words)
51 ...	indices for named banks
JW(14) ...	named bank
	named bank
	...
	named bank
JW(15) ...	free space (gap)
JW(16) ...	work bank
	work bank
	...
... JW(12)	work bank

Named banks

Named banks are stored in the low index part of an array. They have assigned a name, which is a character string of four characters, and a number, which is an arbitrary integer (positive or negative or 0). A bank with a name NAME, a number NR and ND data words is created in the basic array by the statement:

```
IND = NBANK(NAME,NR,ND)
```

The index IND points to the bank, more precisely: the value of IND is the index within the array, which contains the number of data words. The structure of a named bank is shown below.

index	content
IND − 3	name NAME
IND − 2	number NR
IND − 1	index of next bank
IND	number of data words
IND + 1	data word 1
IND + 2	data word 2
...	...
IND + ND	data word ND

According to the definition of the index IND, words of the data area are referenced by statements of the form

```
VALUE = IW(IND+1)+IW(IND+2)
```

and data are stored in banks by statements of the form

```
IW(IND+3) = IVALUE
```

The name and the number of a named bank can be used to find the index of the bank in any program part; for the basic array the statement

```
IND = NLINK(NAME,NR)
```

can be used to determine the index IND. A single bank can be dropped by

```
IND = NDROP(NAME,NR)
```

Larger data structures

By list of names larger data structures can be defined. This concept allows to perform operations on sets of banks, which consist of all banks with names, which are in the list. Lists may be specified by a single character, for example 'E', and there is a subprogram BLIST to edit a given list (add, delete). An important operation for sets of banks is output and input (I/O). A larger data structure can be written to an output medium by one call, in which the list is specified. Input subprograms allow to read and store the data structure. I/O is possible in a fast, machine dependent way, and in a machine independent way, which allows the exchange of data between different types of computer with automatic conversion of data words (integer, floating point, text). This concept is based on the format of banks, which can be specified in a way similar to the FORTRAN FORMAT statement. There is no automatic garbage collection in the area of named banks.

Work banks

Work banks are stored in the high index part of the array. In contrast to named banks no larger data structures can be defined, no standard I/O exists and garbage collections are done automatically, if necessary. Each work bank with ND data words is created with an individual identifier, with a statement

```
CALL WBANK(IW,ID,ND,*)
```

The individual identifier ID is used as an index and has to be in a common, to avoid potential problems with certain optimizing compilers.

The structure of a work bank is given below.

index	content
ID − 3	name (default is 'WORK')
ID − 2	total nr of words in the bank NT
ID − 1	pointer to index ID
ID	number of data words ND
ID + 1	data word 1
ID + 2	data word 2
...	...
ID + ND	data word ND

Work banks are useful and efficient for certain applications like shortliving data banks inside a program module. Their use for the transmission of data between different program modules however tends to decrease the degree of modularity. The system provides *no* subprograms for the input/output of work banks.

2 Initialization and Utility Programs

2.1 Initialization

All BOS-arrays have to be initialized. This is usually done in the MAIN program and preceded by the declaration of the arrays. The basic array IW in COMMON/BCS/ has to be declared with at least 1000 words, and it has to be initialized as the first array, even if it is not used directly afterwards. Up to 10 arrays may be used, however it is recommended to prefer the basic array, since the use of many arrays may contradict the concept of modularity. For histogram like data banks it is recommended to use the array KW in COMMON/HCS/.

Arguments

```
IW   = basic BOS array
JW   = any BOS array
NDIM = dimension parameter of array
```

Declaration and subprogram calls

```
COMMON/BCS/IW(NDIM)....................basic array          (necessary)
COMMON/any/JW(NDIMJ)...................further array         (optional)
COMMON/HCS/KW(NDIMK)...................further array         (optional)

CALL BOS(IW,NDIM)......................initialize basic array(necessary)
CALL BOS(JW,NDIMJ).....................initialize array JW    (optional)
CALL BOS(KW,NDIMK).....................initialize array KW    (optional)
```

By a call of BOS an BOS-array is initialized. Further calls of BOS with the same array as argument are ignored for the basic array; other BOS-arrays are reset to the initial state, using the given dimension parameter or the value at the previous call, if the dimension parameter in the call is zero.

Example: The code below shows the initialization of the basic array with 50000 words and of the array in common /HCS/ with 10000 words.

```
      COMMON/BCS/IW(50000)
      REAL       RW(50000)
      EQUIVALENCE (IW(1),RW(1))
      COMMON/HCS/KW(10000)
      REAL       XW(10000)
      EQUIVALENCE (KW(1),XW(1))

      CALL BOS(IW,50000)
      CALL BOS(KW,10000)
```

2.2 General rules

Arguments

Most of the subprograms act on a specific array, which may be either the basic array IW in common /BCS/ or any BOS-array JW. If the array has to be specified in the call, it is the first argument. Further arguments are either of type integer or of type character string. These arguments are never modified by a call, with one exception: the index ID of a work bank. If a subprogram has to return some value, it is defined as a *function* subprogram, and the returned value is zero, if the required operation could not be performed. A test on the function value should follow a function call, when indicated in the description of the function call. The meaning of the arguments is explained in each section together with the calls. In addition table 6 gives a complete overview over all possible arguments.

Extended arguments

Some subprograms in the BOS system have rarely used additional options, which are *not* selected by an argument; instead the user has to select such an option by setting the first word of the array, JW(1), to a certain value. This word is reset to zero by a call to a BOS system subprogram (JW(1) = 0 means default option).

Error returns and return codes

Error conditions, meaning that the subprogram could not execute the required operation, are indicated either by

- taking the alternate return in cases of subroutines, or by
- a returned value of zero in case of functions.

In both cases the reason is indicated by the value of the second word of the BOS-array, JW(2), if necessary. This return code is explained in each section together with the call (see also table 8). One design criteria of BOS was to minimize the number of cases, where an error return is posssible. For some errors, where a continuation is impossible (for example, if the bank structure is destroyed), the program will stop after printing the reason.

2.3 Conversion functions

Since in FORTRAN77 character strings cannot be stored together with numerical data in the same common, the BOS systems includes two functions for the conversion between the representation as a character*4 string and the representation as packed integer. In the latter representation the four characters are stored in their internal character codes from the left to the right in one machine word. The two functions are machine independent; they use functions from the CERN program library, which exist for almost all types of computers. On some machines the two representations are identical and the functions trivial.

Arguments

```
CHST = character*4 string
INST = integer
```

Subprogram calls

```
CHARACTER*4 CHST,CHAINT
INST = INTCHA(CHST)....................convert from character to integer

CHST = CHAINT(INST)....................convert from integer to character
```

When a named bank is created, the name NAME of the bank is stored in the integer representation within the bank at index IND-3. For the conversion back to a character string, the function CHAINT should be used. The functions should also be used, if text (character strings) is stored in the data part of a bank.

Example. The name of a bank, stored in the integer representation within a named bank at the index IND-3, is converted to the character representation.

```
      CHARACTER*4 CHAINT,NAME
      NAME = CHAINT(IW(IND-3))         conversion to character*4
```

Example. Text from a character string TEXT is stored in the data part of a bank with an index IND. The function INTCHA is used to convert at each call four characters from the string into the integer representation. The FORTRAN77 function LEN and the substring feature is used.

```
      CHARACTER*55 TEXT

      LA=1
      DO 10 I=1,(LEN(TEXT)+3)/4        determine nr of words from length
      LB=MIN(LA+3,LEN(TEXT))
      IW(IND+I)=INTCHA(TEXT(LA:LB))
   10 LA=LB+1
```

2.4 Utility subprograms for printing

The system contains some subprograms to print certain data of the arrays. The call of BOSTA is recommended after initialization of all arrays and at the end of the program.

Subprogram calls

```
CALL BOSTA............................print current status of all BOS
                                      arrays including bank formats
                                      and I/O status

CALL BOSFM............................print bank formats

CALL BOSIO............................print current status of I/O

CALL BOSBL(JW)........................print I/O statistic on banks in
                                      array JW

CALL BOSBK(JW)........................print list of current banks in
                                      array JW

CALL BOSDP(JW)........................print dump of array JW
```

3 Named banks

Named banks are the basic elements of the BOS system and should be used for long-living data. Their properties allow a modular structure of big and complex programs. The structure of named banks is shown in table 1. The system includes subprograms for various operations with single banks and sets of named banks including input/output.

index	content
IND − 3	name NAME
IND − 2	number NR
IND − 1	index of next bank
IND	number of data words
IND + 1	data word 1
IND + 2	data word 2
...	...
IND + ND	data word ND

Table 1: Structure of named banks

3.1 Operations with single named banks

Operations with single named banks require as arguments the name and the number of the named bank. They are performed by functions, which start with the character 'N' for banks in the basic array, and with the character 'M' for an arbitrary BOS-array. Immediately after the function call a test on IND=0 should follow in order to check the success of the operation.

Arguments

```
NAME = name of bank (character*4)
NR   = number of bank
ND   = number of data words of a bank (not negative)
IND  = index of a named bank
```

Any character*4 string may be used as a name, except a string with first character +. As number of a bank any integer is allowed. The number of data words may not be negative.

Subprogram calls

```
IND = NLINK(NAME,NR)
IND = MLINK(JW,NAME,NR)................determine index of named bank
IF(IND.EQ.0) THEN ...
```

Functions NLINK and MLINK determine the index IND of a named bank. The returned value is zero, if the bank does not exist; therefore a test on IND=0 should follow. Other methods for the determination of the index of a bank are discussed in chapter 3.2.

```
IND = NBANK(NAME,NR,ND)
IND = MBANK(JW,NAME,NR,ND).............create named bank with ND data words
                                       or change length to ND data words
IF(IND.EQ.0) THEN ...
```

Functions NBANK and MBANK are used to create a named bank with ND data words and to return the index of the new bank. If the bank exists already, the number of data words is changed to the value given in the argument. All additional words or all words in the case of a new bank are set to zero. If space is insufficient (even after a garbage collection in the work bank area) to create a new bank or to increase the number of data words, the returned value is zero and the return code JW(2) is set to 1 or 2; in this case a new bank is not created and an existing bank is left unchanged (note that in the latter case the bank will exist after return, the returned index however is zero). Programming methods for banks with a varying number of data words are discussed in chapter 7.

Options: For special requirements the following options can be selected:

```
JW(1) = 1   ...........................additional bank words left
                                       unchanged

JW(1) = 2   ...........................move bank to gap of free space

JW(1) = 3   ...........................all other banks of the same name
                                       are deleted
```

If JW(1)=1 is set before the call, the additional data words are left unchanged. If JW(1)=2 is set before NBANK or MBANK is called, the bank is moved to the gap, thus the moved bank will be immediately before the gap. The option JW(1) = 3 allows to create a new bank, deleting all existing banks of the same name before. Since use is made of the space of existing banks, this option will be efficient for very large banks, occupying a large fraction of the total array.

```
IND = NDROP(NAME,NR)
IND = MDROP(JW,NAME,NR)................drop named bank
```

The named bank is dropped. Although it may still be in the BOS array, its index cannot be obtained by NLINK/MLINK and the bank will vanish at the next garbage collection.

```
IND = NPRNT(NAME,NR)
IND = MPRNT(JW,NAME,NR)................print named bank
```

The named bank is printed. If a format is defined for the bank name, it is used.

3.2 Indices for named banks

The standard method to determine the index of a given bank is to use the functions NLINK and MLINK. Although these functions are fast (one call is equivalent to about two calls of a dummy function), there are certain applications with loops etc, where the total time spent in the functions may become large. The algorithm for indices of named banks is explained below and it will be shown, how function calls can be avoided.

Name-indices

To each name appearing in a call of BOS system programs a name-index with values 51, 52 ... is assigned. This assignment is made at the first appearance of a name. The name-index for a given name NAME can be obtained by the statement

```
      NAMI = NAMIND(NAME)            return name-index for name NAME
```

If banks are existing for a given name, the index of the first bank (first means lowest number) is stored in word JW(NAMI). Thus having determined the name-index NAMI for a given name by the above statement, the statement

```
      IND = JW(NAMI)                 index of first bank
```

directly yields without a function call the index of the first bank, or zero, if no bank of the given name exists. During the execution of a program the name-index will never change. Therefore within a subprogram the statements

```
      DATA NAMI/0/                   initialize with zero
      IF(NAMI.EQ.0)NAMI=NAMIND(NAME) define NAMI at first call
```

can be used to determine the name-index at the first call of the subprogram, in order to save time.

The word JW(IND-1) in each bank contains the index of the next bank (in increasing order of bank numbers) of the same name, or zero, if no further bank exists. Thus by the statement

```
      IND = JW(IND-1)                next index
      IF(IND.EQ.0) ...               test IND = 0
```

the next index can be determined without a function call.

Loops

If a loop has to be executed over all banks of a given name (in order of increasing number) one has to initialize the index IND before the loop to NAMIND(NAME)+1 or NAMI+1; the following statements can then be used for the loop:

```
      IND = NAMI+1                   initialize index IND
   10 IND = JW(IND-1)                next index
      IF(IND.NE.0) THEN              test termination of the loop
         ...                         IND is index for the bank with
         ...                         number JW(IND-2)
         GOTO 10
      END IF
```

Note that a loop can be executed without any function call for the determination of indices.

Access to banks in random order

In some applications banks of certain names are required in some (random) order, determined by data and program flow. A function call (NLINK, MLINK) would be possible, but requires of course some time. A faster method in this case is to use subroutine BWIND, which creates a work bank containing all indices of named banks for a given name with numbers between 1 and an upper limit (argument NRMAX of BWIND). This subroutine is applicable in cases, where the bank numbers are (not too large) positive numbers.

```
CALL BWIND(JW,NAME,NRMAX,ID)..........create/modify work bank with indices of
                                      named banks
```

After the call the content of the work bank at index ID will be: ND will be the largest bank number of an existing bank $\leq$ NRMAX. In practice NRMAX should be set to some large number. For example if bank numbers are expected in the range 1 ... 50, one can use NRMAX = 1000. The actual length of the work bank will be the largest number of an existing bank. The work bank contains the actual values of the indices at the time BWIND was called. If new banks are created or existing banks are dropped, the content is not changed. The use of BWIND is recommended in cases, where correlations between banks with a different name are used.

Example: Assume banks with the names NAM1 and NAM2; a bank with the name NAM1 can be correlated to a bank with the name NAM2 and vice versa. The corresponding number of the correlated bank may be stored by the user for example in both cases in the first data word. The code allows the fast access to banks of two names in a random order, only two calls of BWIND are necessary.

index	content
ID	ND
ID+1	index of bank (NAME,1) (or zero)
ID+2	index of bank (NAME,2) (or zero)
...	...
ID+I	index of bank (NAME,I) (or zero)
...	...
ID+ND	index of bank (NAME,ND)

Table 2: Content of work bank created by BWIND

```
DATA ID1/0/,ID2/0/
CALL BWIND(JW,NAM1,1000,ID1)
CALL BWIND(JW,NAM2,1000,ID2)
I1 = ...                            bank number of bank NAM1
IND1=JW(ID1+I1)                     index of bank (NAM1,I1)
IF(IND1.EQ.0) ...                   test index
I2=JW(IND1+1)                       bank number of bank NAM2
IND2=JW(ID2+I2)                     index of bank (NAM2,I2)
IF(IND2.EQ.0) ...                   test index
```

3.3 Bank formats

The format of a bank describes the type of data stored in the bank. All banks of the same name have to be described by the same format. This requirement of course has consequences for the design of banks. The definition of a format is essential for machine-independent input/output, it is useful for the printing of banks and if a bank is copied to a work bank. In machine-independent output, the data words are converted according to the format to a standard 32-bit representation, and in machine-independent input the data words are converted to the internal machine representation. It is recommended, not to use complicated formats (although it is possible), and to prefer the floating-point format.

The format of a bank can be either *mixed* or *bit-packed*. The type of the data words is described by a character string FMT.

Mixed format

The format desription FMT follows the same rules as FORTRAN-format statements, except that no length information is specified:

I	integer
F	floating-point
A	text (4 characters per word)

The character string defining the format is of the type

$$'f_1, f_2, \ldots' ,$$

where the f_i are format codes nI, nF, nA or $n(\ldots)$ for a group format specification (n = constant), for example:

'I,5F,2(I,3F)'

Outer parentheses of the format can be omitted. Note that, according to the FORTRAN rules, when format control reaches the last (outer) right parenthesis and there are data words left, the format starts again by the last preceding right parenthesis, including its group repeat count, if any, or, if no group specification exists, then at the first left parenthesis of the format specification.
If a bank contains only text, the use of the format '18A' is recommended, because in this case the printing routines use a format without gaps.

Bit-packed format

The 32 rightmost bits of each machine word are used, they contain either 32-bit words or two 16-bit words or four 8-bit words. The format description is:

'B32'	for 32-bit words
'B16'	for two 16-bit words
'B08'	for four 8-bit words

In machine-independent output the content (32 rightmost bits) of the machine words are transmitted without conversion. In machine-independent input, the data are transmitted to the 32 rightmost bits of each machine word. If the named bank is copied to a work bank (BKTOW), the data are unpacked according to the format (expansion). If a work bank is copied to a named bank (BKFRW) and the format for the name is bit-packed, the data are packed (compression). In printing named banks with a bit-packed format the content of the banks are printed unpacked.

Arguments

```
NAME   = name of bank
FMT    = character string for mixed format
       = 'B32' or 'B16' or 'B08' for bit-packed format
```

Subroutine call

```
CALL BKFMT(NAME,FMT)..................format FMT is assigned to all banks with
                                      the name NAME
```

The bank format for the given name is defined. The call is ignored, if the format has already been defined by a call. On output the format descriptions are added automatically to the records, in a subsequent input the format description are read. Thus formats are also defined by input of records. However, the first call of BKFMT overwrites a format defined from a record. If the format description in a call has a syntax error, the program will stop at the first usage of the erroneous format description.

Examples: The interpretation of several format strings is given.

```
                                      format of data words:

CALL BKFMT('HEAD','2I,3F')            IIFFFIIFFFIIFFF...   (IIFFF repeated)

CALL BKFMT('DATA','F')                FFFFF...  (all floating-point)

CALL BKFMT('DATA','(F)')              FFFFF...  (all floating-point)

CALL BKFMT('TRAC','2I,A,2(2I,F),(F)') IIAIIFIIFFFFFF...

CALL BKFMT('RAWD','B16')              16-bit packed
```

3.4 Sets of banks

For various operations (input/output, printing etc.) it is necessary to define a set of banks, which belong together. In BOS a set of banks is defined by the list of all names of banks, which belong to the set. This is simple, but not completely general: the case where some banks of a given name are included in the set of banks and others of the same name are not included in the set, cannot be described. If, in any operation, the list contains names of banks, which are not existing, these names are ignored.

A list of names may be an internal list or an explicit string of names, for example 'HEADTRACLIST' for the names HEAD, TRAC and LIST. Internal lists are specified by a single character L; the character can have one of the values

C E R S T and 0 (empty list)

The names of internal lists are stored in the BOS array. The subroutine BLIST can be used to edit an internal list (add names, delete names). The function NLIST can be used to obtain the n-th name of a list.

Arguments

```
OPT    = 'L='   set internal list L to LIST; L = C E R S or T
       = 'L+'   add list LIST to internal list L
       = 'L-'   delete list LIST from internal list L
LIST   = explicit string of names
       = 'L' character, specifying the internal list; L = C E R S T 0
N      = index within a list
```

Subprogram calls

```
CALL BLIST(JW,OPT,LIST)................change internal list according to option
                                       OPT using list LIST
```

The internal list specified within OPT is either set equal to the list LIST ('L='), or the names from list LIST are added to or deleted from the internal list ('L+' or 'L-')

```
CHARACTER*4 NLIST
NAME = NLIST(JW,N,LIST)................return N-th name from list LIST or blank
```

The name of the N-th entry in the list LIST is returned. The value blank (NAME = ' ') is returned, if $N < 1$ or $N >$ the number of names in the list.

```
CALL BDROP(JW,LIST)....................drop set of banks defined by list LIST
```

All banks with names in the list LIST are dropped.

```
CALL BPRNT(JW,LIST)....................print set of banks defined by list LIST
```

All banks with names in the list LIST are printed.

Examples: A few calls of BLIST are shown, together with the content of the internal list E after each call.

```
                                       content of list E after call

CALL BLIST(JW,'E=','HEADRAWD')         HEAD RAWD

CALL BLIST(JW,'R=','SHOW')

CALL BLIST(JW,'E+','R')                HEAD RAWD SHOW

CALL BLIST(JW,'E+','TRAC')             HEAD RAWD SHOW TRAC

CALL BLIST(JW,'E-','SHOW')             HEAD RAWD TRAC

CALL BLIST(JW,'E+','TRAC')             HEAD RAWD TRAC

NAME = NLIST(JW,2,'E')                 returned function value is NAME='RAWD'

CALL BLIST(JW,'E=','0')                empty
```

3.5 Changing the names of banks

The name and number of a named bank can be changed by subprograms NSWAP/MSWAP and BSWAP.

Arguments

```
NAM1, NAM2 = names of banks
NR1, NR2   = numbers of banks
```

Subprogram calls

```
IND = NSWAP(NAM1,NR1,NAM2,NR2)
IND = MSWAP(JW,NAM1,NR1,NAM2,NR2)......change name and number
```

If both banks are existing, the bank (NAM1,NR1) will get name NAM2 and number NR2, and the bank (NAM2,NR2) will get the name NAM1 and the number NR1; the index of the (new) bank (NAM1,NR1) is returned. If only one of the banks exists, its name and number is changed to the other name and number, and its index is returned.

```
CALL BSWAP(JW,NAM1,NAM2)...............exchange names NAM1 and NAM2
```

All banks names NAM1 are changed to NAM2 and all bank names NAM2 are changed to NAM1. If banks of one of the names do not exist, the banks of the other name are renamed. The order of the two names in the argument is irrelevant.

Example: The names 'TRAC' and 'SPUR' are exchanged two times. After the two calls the banks have the old names again.

```
      CALL BSWAP(JW,'TRAC','SPUR')     exchange names
      CALL BSWAP(JW,'TRAC','SPUR')     change names back
```

3.6 Garbage collection

Garbage collection in the area of named banks regains the space occupied by dropped banks. The position of remaining banks may change by this operation. Garbage collection for the area of named banks is not done in any system subprogram (with the exceptions of BDASQ and BDALD and also the general input-output program BSQER) and has to be called explicitly by the user. In typical applications this is done by the user in regular intervals, for example before a new record is read in.

```
CALL BGARB(JW).........................garbage collection for named bank area
```

4 Work banks

Work banks are created with an individual index ID. They should be used preferentially inside a program module and *not* for the transmission of data between modules, since their properties may contradict the requirements of modularity and no input/output is foreseen. They are however useful and efficient inside a module; they do not disturb the area of named banks and allow a garbage collection with automatic update of all indices of work banks. The structure of a work bank is given below. Garbage collection in the region of the work banks is done automatically, if neccessary, with update of all individual indices of existing works banks

index	content
ID − 3	name (default is 'WORK')
ID − 2	total number of words in the bank NT
ID − 1	pointer to index ID
ID	number of data words ND
ID + 1	data word 1
ID + 2	data word 2
...	...
ID + ND	data word ND

Table 3: Structure of a work bank

When a work bank is created, the index of the work bank is stored in ID, and the address of the word ID is stored in the work bank at JW(ID-1). This allows the system to update the content of indices ID automatically during a garbage collection. This method requires, that the content of the work bank is referenced directly with the index ID, which was used during the creation of the bank. If the argument ID is nonzero at entry to a work bank subroutine, it is checked: it should point to a work bank, and word JW(ID-1) should contain the address of the word ID. If the value of ID is incorrect, the program will stop. Before a work bank is created, the index ID

must have the value zero. The value of an index should *never* be changed by the user directly. The indices ID of all work banks should be in (labelled) commons; otherwise there are potential problems with certain optimizing compilers (index ID not updated during a garbage collection).
Work bank indices or the work bank data part should not be an argument in a subprogram, if the subprogram itself may create or drop a work bank.

4.1 Operations with work banks

Arguments

```
ID     = index of the work bank
ND     = number of data words of the work bank (not negative)
```

Subprogram calls

```
DATA ID/0/
CALL WBANK(JW,ID,ND,*).................create work bank
                                       * insufficient space
```

A work bank with ND data words is created, the index is stored in ID. If the bank already exists (ID $\neq 0$ at entry), the number of data words will be changed to the value of ND given in the argument. For a successfull operation, the normal return is taken with index ID defined.
The alternate return is taken in case of insufficient space; if space is insufficient (even after a garbage collection in the work bank area) to create a new bank or to increase the number of data words, the return code JW(2) is set to 1 or 2 and in this case a new bank is not created and an existing bank is left unchanged.

```
CALL WDROP(JW,ID).....................drop work bank
```

The work bank at index ID is dropped and ID is set to zero. If ID = 0 at entry, immediate return is taken.

```
CALL WPRNT(JW,ID).....................print work bank
```

The work bank at index ID is printed. As an exception the index may also be an index of a named bank.

```
CALL WSWAP(JW,ID1,ID2).................exchange indices for work banks
```

The indices ID1 and ID2 of the banks are exchanged. If one of the indices is zero, the index is transferred.

A garbage collection in the area of work banks is done automatically, when necessary. The user never has to call the garbage collection directly, but for completeness the call is given.

```
CALL WGARB(JW).........................garbage collection for work banks
```

Example: A work bank with 1000 words is created and used. After the use the work bank is printed and dropped.

```
DATA ID/0/
CALL WBANK(JW,ID,1000,*100)
JW(ID+1)=31415
...
CALL WPRNT(JW,ID)
CALL WDROP(JW,ID)
```

4.2 Bank copy to or from work banks

A named bank in an array JW1 can be copied to a work bank in array JW2 and vice versa. If a format of the type 'bit packed' is specified for the named bank and a copy is done to a work bank, the work bank will contain the data unpacked into single words. Conversely, if a work bank is copied to a named bank and the format of the named bank is specified as 'bit packed', the work bank is assumed to contain unpacked data and the named bank will contain the data in packed form.

Arguments

```
JW1    = BOS array of named bank
NAME,NR= name and number of bank

JW2    = BOS array of work bank
ID     = index of work bank
```

Subprogram calls

```
CALL BKTOW(JW1,NAME,NR,JW2,ID,*).......copy named bank (source) to
                                       work bank (target)
                                       * named bank not existing

CALL BKFRW(JW1,NAME,NR,JW2,ID,*).......copy from work bank (source)
                                       to named bank (target)
                                       * work bank not existing
```

If the target bank is existing before the call, the content will be replaced. If the source bank does not exist, the alternate return is taken.

5 Input/Output (sequential)

This chapter explains the subprograms for the input/output of sets of named banks, which are defined by lists of names. There are two posiibilities for the input/output:

- a standard mode selected by the option 'FORT', which is fast and recommended, if a data set is read several times on the same machine; it is based on standard FORTRAN I/O; all records have the same fixed length;
- a machine-independent mode selected by the option 'EPIO', based on the CERN EPIO package [1]; this mode includes data conversion according to the format description of the banks; in writing, floating point and integer data are converted to the IBM representation, character data are converted to ASCII code; in reading, the words are converted back to internal machine representation; if the format is bit packed, the rightmost 32 bits are written and read without conversion.

In both modes a buffer is used and existing format description of banks are automatically written and therefore available in subsequent programs reading the data set.
For each unit LUN the mode is selected by a call of subprogram BUNIT.

Arguments

```
LUN    = unit number
IOMODE = mode of I/O, either 'FORT' or 'EPIO' (character*4)
NBLK   = number of data words for output record ( in words for 'FORT'
         and in 16-bit units for 'EPIO')

CALL BUNIT(LUN,IOMODE,NBLK)............define mode for I/O and buffer
                                       length
```

The argument IOMODE is of type character*4 and may be either 'FORT' or 'EPIO'; any other string is assumed to be 'FORT'. The argument NBLK specifies the buffer size (and record length) for the output of BOS-records. The argument NBLK is in machine words for the option 'FORT' and in 16-bit unit for the option 'EPIO' and has to be a multiple of 180 in the latter case. For input the buffer size is taken from the actual record length, the value of argument NBLK is irrelevant and may be zero. If the argument NBLK is zero for writing, a default value is assumed. If BUNIT is not called for a unit, the mode 'FORT' is assumed with the default buffer size.

Subprograms for the input/output of BOS-records are described in the next section. A general input/output subprogram BSEQR exists, which provides a simple interface to the input/output subprograms and will be sufficiently general for most applications. It also includes the necessary calls for the transition from one record to

[1] H.Grote and I. McLaren, EPIO manual, DD/EE/81-2, CERN Computer Center Program Library I 101

the next record (dropping of banks of the previous records, garbage collection). The use of subroutine BSEQR is explained in chapter 5.2 and a newcomer to BOS reading this manual may directly skip to this chapter.

5.1 Input/output subprograms

Arguments

```
LUN    = unit number
LIST   = list of names
```

By calling

```
CALL BREAD(JW,LUN,LIST,*1,*2)..........read set of banks from unit LUN
                                       *1 read error
                                       *2 end-of-data
```

the next BOS-record is read and the banks are stored in array JW. The list LIST in this call may be only of the one-character form. At return the list contains the names of the banks read in. In addition the word JW(3) is set to the index of the first bank of the record at a normal return. Alternate returns are taken in case of read errors and the end-of-data condition. At the next call of BREAD after a read error the wrong record is skipped and the next record is read. By calling

```
CALL BWRITE(JW,LUN,LIST)..............write set of banks to unit LUN
```

all banks specified in the list LIST are written; if JW(1) is set to 1 before the call of BWRITE, the banks will also be dropped. Since a buffer is used, it is *important* to write finally the last buffer, which is done using an empty list (LIST = '0'),

```
      CALL BWRITE(JW,LUN,'0')
```

or by a rewind of the data set using subroutine BRWND. A single bank (NAME,NR) is written by the call

```
CALL BWRSB(JW,LUN,NAME,NR).............write single bank to unit LUN
```

in one BOS-record.
The data set LUN is rewound by the call

```
CALL BRWND(LUN).........................rewind data set unit LUN
```

This includes the output of the last record in the buffer (for output). After a rewind a data set previously used for writing may be used for reading.

Example: The following code shows the statements for a sequence of operations, in which a record is read, processed and written. After writing all banks of the record are dropped, and after a garbage the process is repeated. The basic array IW and the list 'E' is used.

```
   10 CALL BREAD(IW,1,'E',*10,*100)
      ...(processing)
      CALL BWRITE(IW,2,'E')
      CALL BDROP(IW,'E')
      CALL BGARB(IW)
      GOTO 10
  100 CALL BWRITE(IW,2,'0')
      END
```

Special read option: A special option of BREAD allows a preliminary read of a record. After a call of BREAD with argument LIST = ' ', the user has access to the first bank of the new record, although this bank is not stored as an ordinary named bank and cannot be found by the functions NLINK/MLINK. The user may then decide either to accept the record or to read the next record. If in a certain application only selected records are required, the execution time of the program will be shorter using this option.

```
CALL BREAD(JW,LUN,' ',*,*)............preliminary read
                                      *1 read error
                                      *2 end-of-data
```

The next record is read preliminary. At a normal return JW(3) contains the index of the first bank of the record. Alternate returns are taken in case of read errors and the end-of-data condition.
At a next call of BREAD with the same unit LUN, the same BOS array JW and the usual one-character form of the argument LIST the banks of the preliminary read record are stored. If the next call has as third argument again LIST = ' ', the next record will be read preliminary (ignoring the previous record).

Example: Assume that records from unit 1 are required only, if the first bank (HEAD,0) contains the value 13 in its first word. Without the special option this could be done with the following code.

```
   10 CALL BREAD(IW,1,'E',*20,*30)
      IND = NLINK('HEAD',0)
      IF(IND.EQ.0)        GOTO 15
      IF(IW(IND+1).NE.13) GOTO 15
      ...
   15 CALL BDROP(IW,'E')
      CALL BGARB(IW)
      GOTO 10
```

With the special option the code has two more statements, but the execution time will be shorter.

```
   10 CALL BREAD(IW,1,' ',*20,*30)
      IND = IW(3)
      IF(IW(IND-3).NE.INTCHA('HEAD')) GOTO 10
      IF(IW(IND-2).NE.0)              GOTO 10
      IF(IW(IND+1).NE.13)             GOTO 10
      CALL BREAD(IW,1,'E',*20,*30)
      ...
      CALL BDROP(IW,'E')
      CALL BGARB(IW)
      GOTO 10
```

Job control statements

Job control statements are given for the IBM MVS system of the DESY computer center. For the input/output in the 'FORT' mode the DD statement has to specify as usual

```
//FTXXF001 DD DSN=...
```

for the unit LUN = XX.
The IBM version of the EPIO package uses the CERN IOPACK package [2] and the rules on JCL explained in the IOPACK Users Guide apply for the mode 'EPIO'. At DESY the DD-name of a file on unit XX is IOFILEXX, the recordformat is U (see IBM DD-card example below) with a blocksize BLKSIZE = 2 × NBLK:

```
//G.IOFILEXX DD DSN=...,DCB=(RECFM=U,BLKSIZE=...),...
```

5.2 General input/output subprogram

The general subprogram for sequential input/output BSEQR contains all calls to read a data set sequentially, to write the records on a sequential data set (optionally), and also to drop the banks of the previous record and to make a garbage collection before a new record is read. It may either be used directly or serve as a model for a similar, but specialized program for a certain application. The following conventions apply;

- The basic array IW/RW in common /BCS/ is used to store the banks of the read records.
- The list 'E' is used to keep the names of the banks.

[2] CERN, IOPACK Users Guide, R. Matthews, CERN Computer Program Library Z300

- Records with read errors are skipped.

BSEQR allows to use one input unit and up to four output units. The mode for each of these units has to be selected by the user *before* the first call of BSEQR by a call of BUNIT. An example for the minimal version of a MAIN program for reading a data set and processing (assumed to be done in subprogram MODULE) is given below.

```
      COMMON/BCS/IW(5000)
      CALL   BOS(IW,5000)
      LUNR = 1
   10 CALL BSEQR(LUNR,IRET)
      IF(IRET.LT.0) GOTO 100
*     ...
      CALL MODULE
*     ...
      GOTO 10
  100 STOP
      END
```

In this MAIN program all records from unit 1 are read, after each input of a record the subprogram MODULE is called for processing.
The subprogram BSEQR has several options, described below.

Input

`CALL BSEQR(LUNR,IRET)..................read from unit LUNR`

The argument LUNR specifies the unit for input, and may have the values 1 or 2 or 0. The value 0 means no input, the list 'E' is reset to contain no bank names; this option may be used, if all banks are created during processing. The value of the argument LUNR is relevant only at the first call of BSEQR and ignored afterwards. The return argument IRET has the value 1, if a record has been read. If the end-condition is reached in BSEQR (for example by an end-of-file on the input unit), the argument IRET is 0 at return and will be -1, if BSEQR is called once more. The IF-statement in the MAIN program above assumes one pass through the processing module after the end-condition. This will not be done if the test is for IRET.LE.0.

Parameter

By default BSEQR will stop reading 2 seconds before the timelimit or at an end-of-file on the input data set. These limits are defined by parameters, which can be changed by a call of BSEQP before the first call of BSEQR.

`CALL BSEQP(NSEC,NLIM)..................define parameter`

Reading will stop NSEC seconds before the time limit or after NLIM records, where NLIM = 0 means no limit. If the user wants to stop reading at a certain condition, he may call BSEQE at any time.

`CALL BSEQE............................require end condition`

At the next call of BSEQR the end-condition is assumed.

Output

Default is no output. Output can be done to up to four different units. The mode of output has to be defined for each unit by a call to BUNIT before the first call of BSEQR (default mode is 'FORT'). A call to BSEQW sets an output flag; actual output is done at the next call of BSEQR. If the end-condition is reached in BSEQR, the buffers for used output units will be written.

`CALL BSEQW(LUNW)......................set output flag`

If LUNW is given negative, the output to the unit is suppressed. Calls with different values of the argument can be in any order. The last value of the output flag for all units is relevant.

Further options

The subprogram BSEQR contains further options, which are described in section 10.

6 Card image input

6.1 Data banks

Banks may be read from card image records (or cards) in free format (cols. 1 - 72 are used). The data for one bank may be on one or several card image records. The following conventions apply:

- integers: string of decimal digits, containing no decimal point and no blank, optionally signed.
- real numbers: string of decimal digits with a decimal point and without blank, optionally signed and optionally followed by exponent with a letter E or D and an integer.
- text data: string of characters (except apostroph), enclosed in two apostrophs on the same card image record (stored with 4 characters per word).
- label: the characters of the four columns 1 ...4 before the first digit or any of the characters + or -. If the label of a cardimage record is blank, the record is considered as a continuation records. If the label is not blank, it is used as the name of bank, in which the data are stored.
- comment: all characters after an ! or a single apostroph as well as any additional characters are treated as comment and ignored. A card image record with a ∗ in column 1 is treated as comment card.

Any characters between integers, real numbers and text are ignored. The data are stored in a bank with a name equal to the label. The bank number is either given as an integer between the label and a slash (/), or choosen automatically as 1+ the highest existing number of a bank of the given name, or zero, if no bank exists.

Special labels

The following card image records have a special meaning and are not stored.

```
UNIT  integer     reading is switched to the given unit number (and is
                  switched back to the system input unit if EOF found)

POFF              printout of records is switched off

PON               printout of records is switched on (default)

ENDQ              last record (forcing return of the reading program)
```

The content of the card image records is as follows:

```
name   integer, floating point, text data
name   integer, floating point, text data      ! comment
       continuation of data
*      comment card
name nr /  integer, floating point, text data   ! comment
name nr /  integer, floating point, text data
       continuation of data
UNIT  integer
POFF
PON
ENDQ
```

There are two subprograms for reading card image records, a function reading one bank into any BOS array and a subroutine reading a set of banks into the basic BOS array.

```
CALL BREADC...........................Read records (starting from
                                      system input unit) and store
                                      data in banks in the basic array
                                      IW in common /BCS/ until 'ENDQ'-
                                      record or EOF on system input
                                      is found

IND = MREADC(JW)......................one bank is read and stored in array JW
                                      IND = 0 for 'ENDQ'-record or EOF
                                      In case of EOF in addition JW(2)=7
```

Example: The following card image records are read by BREADC, the resulting banks are shown below.

```
DATA 27 / 311  75   3.1    3.1E01    ! COMMENT
* THIS IS A COMMENT
CONS 32 57.0 'EXPERIMENT'  3.145
 25.7   22.2                         ! COMMENT
ENDQ
```

```
            BANK  (DATA,27)            BANK   (CONS,0)
            ---------------            ---------------
            IND       4                IND       8
            ---------------            ---------------
            IND+1     311              IND+1    32
            IND+2     75               IND+2    57.0
            IND+3     3.1              IND+3    'EXPE'
            IND+4     31.0             IND+4    'RIME'
                                       IND+5    'NT  '
                                       IND+6    3.145
                                       IND+7    25.7
                                       IND+8    22.2
```

6.2 Format and text banks

Additional options allow to define bank formats by card image records and to store text from whole card image records (cols. 1 - 72) in banks. These options are selected by the character strings $FMT, $TEXT, $TEXTC and END$. The content of the card image records is as follows.

```
name  $FMT 'bank format'        ! comment
name  nr / $TEXT                ! comment
   text
   ...
   text
END$
name       $TEXT                ! comment
   text
   ...
END$
name  nr / $TEXTC               ! comment
   text
   ...
END$
```

A card image record with the string $FMT directly defines a bank format. Function MREADC will not return after reading this record, but proceed and return with the index of the next bank stored.
After finding the string $TEXT or $TEXTC all following card image records will be read and stored as text, until a label END$ is found. The format of the text bank will automatically defined to be '18A'.

Example:

```
HEAD   $FMT   '8I,2A,(10F,10I)'
RUNC   4711   $TEXT
A special trigger is valid for this run.
Trigger 16 has been switched off.
END$
```

The text option $TEXT allows of course also to read card image records like the ones explained in chapter 6.1 and to store the text in a bank. The following card image records would be stored as text in the bank (BNKS,311):

```
BNKS 311 /   $TEXT
DATA 27 / 311  75   3.1    3.1E01    ! COMMENT
* THIS IS A COMMENT
CONS 32 57.0 'EXPERIMENT'  3.145
 25.7   22.2                         ! COMMENT
END$
```

For banks with text of this kind the subprogram BCALLC allows to interprete the text and to store the banks defined by the text as if they are read by MREADC or BREADC.

```
CALL BCALLC(JW,NAME,NR)................interprete text from bank
```

The bank (NAME,NR) in array JW is assumed to contain text in its data part, which is interpreted like reading card image records and the banks are stored in array JW. A statement

```
      CALL BCALLC(JW,'BNKS',311)
```

would, for the example above, create just the banks of the example in the chapter 6.1. If the character string $TEXTC is used instead of $TEXT, the call of BCALLC for the bank is done automatically after reading the text.

7 Direct access I/O of single banks

7.1 Introduction to direct-access operations

Single banks can be written on a direct-access data set and can be read in a random order. The method used and the basic properties are described briefly.

The standard FORTRAN77 features for direct access are used. A direct-access data set has a fixed number NREC of records and a fixed number IRECL of words per record. About 5 % of the records are used as directory records, which contain for all stored banks name NAME and number NR and the number of the data record of the bank. Directory records can have continuation records. The first three records of a direct-access data set are system records, which contain the status of the data set. The following records are data records, followed by unused records and, finally, by the directory records. There is no limitation on the length of banks; long banks are automatically split to more than one data record.

If a bank (NAME,NR) has to be read, a hash function is used to calculate from the name NAME and the number NR the number of the directory record, from which the number of the data record is obtained. The data record is read and possibly also the next record(s), if the bank is split, and the bank is stored in the BOS array. Usually only two records have to be read. If a bank (NAME,NR) has to be written, the bank is added to the last used data record and possibly to the next record(s), and an entry is made in the directory record. If the bank already exists on the data set, and has the same length, the bank content in the data record(s) is overwritten by the content of the new bank. If the existing bank has a different length, the existing bank is dropped. Dropping a bank means, that the bank is deleted from the data record(s) and from the directory record.

Simultaneous access for read, write and drop is possible by more than one job at a time. In order to allow modification of the data set by more than one job, the data set is temporarily reserved for one job, using the system records.

A value of 1000 ... 2000 for IRECL, optimized for the specific direct-access unit, is recommended. Both NREC and IRECL have to be > 20.

At the initialization of a data set a password can be specified. Operations which modify the data set are possible only if the correct password is given (BDABF), otherwise they are ignored. Read operations are always possible.

7.2 Initialization

A direct-access data set has to be initialized once by the subprogram INITDA, which can run outside BOS. In an initialization the system records and all other records are preset.

Arguments

```
LUN    = unit
IRECL  = number of words/record
NREC   = number of records
PASSWD = password (character*4)
```

Subprogram call

There are two alternatives for the initialization subroutine, as given below. The version INITDA contains an OPEN statement. The version INITDU requires an user OPEN statement, executed by the user before the call; INITDU itself does not contain an OPEN statement.

```
CALL INITDA(LUN,IRECL,NREC,PASSWD).....initialize direct-access data set

CALL INITDU(LUN,IRECL,NREC,PASSWD).....initialize direct-access data set
```

On the DESY IBM MVS system the following DD card is necessary (4*IRECL = constant):

```
//FTXXF001 DD DSN=...,SPACE=(4*IRECL,NREC),UNIT=...,DISP=(NEW,CATLG),
// DCB=(BLKSIZE=4*IRECL)
```

7.3 Open direct access

A direct-access data set has to be opened by a call of subroutine BDABF before any other call of direct-access subprograms, described in the following. If the data set is not opened, all further calls will be ignored. The value of IRECL has to identical to the value, given in the initialization. The subroutine BDABF has been changed since 1986; now it allows an user OPEN statement (sometimes necessary because of strange behaviour of OPEN statements in some compilers). This option and also the selection of shared/exclusive access is now done by an additional argument in IW(1), as explained below.

Arguments

```
LUN    = unit
IRECL  = number of words/record
DSN    = data-set name
PASSWD = password (character*4)
```

Subprogram call

```
IW(1)=IDAOPT
CALL BDABF(LUN,IRECL,DSN,PASSWD).......open direct access data set

IDAOPT = 0   internal open with status 'UNKNOWN'; simultaneous access
             possible; use DISP=SHR in DD card on DESY IBM MVS system
IDAOPT = 1   internal OPEN with status 'OLD'; exclusive access (faster
             for writing, but only 1 user)
IDAOPT = 2   user OPEN statement; simultaneous access assumed
IDAOPT = 3   user OPEN statement; exclusive access assumed
```

7.4 Direct access operations

Arguments

```
LUN     = unit
NAME,NR = name and number of bank
IND     = index of bank
```

Subprogram calls

```
CALL BDAWR(JW,LUN,NAME,NR)............add bank to direct access data
                                      set
CALL BDADR(LUN,NAME,NR)...............drop bank on direct access data
                                      set
IND = MDARD(JW,LUN,NAME,NR)...........read bank from direct access data
                                      set
```

7.5 Search operations

Function NDANR allows search operations for banks existing on a direct-access data set. Defining a bank number NR in the argument, the search can be in the direction of increasing bank number using the argument 'GE' (greater or equal), or in the direction of decreasing bank number using the argument 'LE' (less or equal). A check whether a bank with a given number exists on the data set is possible using the argument 'EQ' (equal). The returned value NNR will be zero, if no bank with the specified condition can be found (therefore it is not possible to search for a bank with number 0).

Arguments

```
LUN  = unit
NAME = name of bank
LEG  = 'LE' : NNR is largest bank nr less than or equal to NR (or zero)
     = 'EQ' : NNR is NR (or zero)
     = 'GE' : NNR is smallest bank nr greater than or equal to NR (or zero)
NR   = number of bank
```

Subprogram call

```
NNR = NDANR(LUN,NAME,LEG,NR)..........search data set for bank number
```

Example: The banks with name 'CCAL' are read in increasing order of bank number, starting at NR = 100.

```
      NR=100-1
   10 NR=NDANR(LUN,'CCAL','GE',NR+1)
      IF(NR.EQ.0) GOTO ...             no further bank on data set
      IND=MDARD(JW,LUN,'CCAL',NR)
      IF(IND.EQ.0) GOTO 10             not enough space to store bank
      ...
      GOTO 10
```

Example: Assume constants are stored in banks with the name 'CCAL', they are valid for a range of data runs NRUN1 ...NRUN2, and are stored on the direct-access data set with the number equal to NRUN1 (the number of the first run, for which they are valid). For a given run number NRUN one has, according to the rule above, to use the bank with the largest number $\leq$ NRUN.

```
      NRUN=...
      NR  =NDANR(LUN,'CCAL','LE',NRUN)
      IF(NR.EQ.0) GOTO ...             error: no bank existing
      JW(1)=1
      IND=MDARD(JW,LUN,'CCAL',NR)      read bank
      IF(IND.EQ.0) GOTO ...            not enough space
      ...
```

7.6 Sequential read

Banks from a direct-access data set can be read sequentially in the order of the data records.

Argument

```
LUN  = unit of direct access data set
```

Subprogram call

```
IND = MDARS(JW,LUN)....................read next bank from data set LUN
```

The next bank from the data set is read, starting with the first bank. For calls after the end-condition reading starts again with the first bank.

Example:

```
   10 IND=MDARS(JW,LUN)
      IF(JW(2).NE.0) GOTO ...          data-set not initialized or END-OF-DATA
      IF(IND.EQ.0) GOTO 10             insufficient space for bank
      NAME=CHAINT(JW(IND-3))
      NR  =JW(IND-2)
      ...
      IND=MDROP(JW,NAME,NR)
      GOTO 10
```

7.7 Printout

Arguments

```
LUN  = unit of direct access data set
NAME = name of bank: all bank numbers are printed in order
     = '    ': all names of banks of the data set are printed
     = '+DIR': the directory records are listed
```

Subprogram call

```
CALL BDAPR(LUN,NAME) ..................Print table of content
```

7.8 Unloading and loading

All banks on a direct-access data set can be copied to a sequential data set (standard) using subprogram BDASQ. In this case the sequential data set will contain BOS-records, each with a single bank. Banks from a sequential data set can be added to a direct access data set using subprogram BDALD. This subroutine uses internally the subprogram BDAWR; thus banks already present on the direct access data set are replaced. A faster version avoiding some overhead present in this version will be prepared in the future.

It is important to note, that both subprograms use the normal storage of named banks during operation. If banks with identical names are present in the BOS array, these may be deleted. It is recommended, to use either an additional BOS array for these subprograms or to use these subprograms in an special job. It is further recommended to select in BDABF the option with exclusive access, if BDALD is used. These programs call the garbage collection subroutine BGARB! The sequential data sets are rewound before return.

Arguments

```
 LUN1 = unit of direct access data set
 LUN2 = unit number of sequential data set
```

Subprogram calls

```
CALL BDASQ(JW,LUN1,LUN2)...............Copy all banks from data set LUN1
                                       to data set LUN2

CALL BDALD(JW,LUN1,LUN2)...............Add all banks to data set LUN1
                                       from data set LUN2
```

8 Changing the length of a bank

Named and work banks are contiguous (without gaps) in storage, and therefore the change of the number of data words of a bank is a distorting operation. However for certain applications this is certainly necessary and should be done in an efficient way, as explained below.

8.1 Work banks

In work banks the word JW(ID−2) contains NT, the total number of words of the bank. If a work bank is created with ND data words using WBANK, then NT=ND+4. The value of NT (and not ND) is used in system operations (shift of banks) as the length of the bank. If by a subsequent call of WBANK with the same index ID the number ND of data words is increased, the system has to shift the work bank and also work banks created after the initial creation of the work bank under consideration. The length NT is updated. If by a subsequent call of WBANK the number ND of data words is reduced, only the value of JW(ID) $\equiv$ ND is changed, while NT is left unchanged. This is a simple and fast operation (which could also be done by the user directly without a subprogram call). If afterwards the number ND of data words is increased again, this is possible without a shift of banks as long as ND $\leq$ NT $-$ 4. If the number ND of data words of a work bank is frequently changed (and the bank is never dropped), the total length will approach the maximum length and there will be only few time-consuming shift operations.

8.2 Named banks

If the number ND of data words of an existing bank is increased by NBANK or MBANK, the bank will be copied to the gap (free space) and increased in length, the bank at the previous position is dropped. If the bank under consideration is the last named bank (just before the gap), no copy is necessary. In the general case however the copy operation is necessary, because a shift (like in the case of work banks) is not possible for named banks (indices of other banks would change). If the number ND of data words of a named bank is reduced by a call of NBANK or MBANK by more than 4 words, this operation is possible without a copy; the systems marks the gap between named banks as if it is dropped bank (when a named bank is dropped, the content of JW(IND) $\equiv$ ND is changed to −ND−4). This is not possible for a decrease by less than 4 words and the system will copy the bank to the gap, as in the case of an increase of ND.
The conclusion from this discussion is the following: the frequent change of the number of data words ND of named banks should be avoided. This can be avoided by using a work bank, which allows frequent changes in an efficient way. If the final number of data words is reached, the bank can be copied to a named bank using subprogram BKFRW.

9 Tables

The structure of BOS arrays is shown in table 4. Each array has system words in the first 50 words, explained in table 5, followed by a region of fixed length for the indices of named banks, and followed by the named banks. Separated by the unused space (gap), the work banks extends to the end of the array. The table 5 explains the systems data in the first 50 words of each array. Words with default value specified can be changed by the user after the initialization. In tables 6, 7 and 8 the arguments of BOS system programs, the extended arguments and the return codes are given.

9.1 Printout

Bank printout and certain messages (for example a nonzero return code) are printed on the unit specified by JW(6). The amount of printout is limited by JW(7) and JW(8):

index	content
1 ... 50	systems data (50 words)
51 ...	indices for named banks
JW(14) ...	named bank
	named bank
	...
	named bank
JW(15) ...	free space (gap)
JW(16) ...	work bank
	work bank
	...
... JW(12)	work bank

Table 4: Structure of an BOS array

word	default value	content
JW(1)	0	additional (input) argument
JW(2)		return code (output) argument
JW(3)		index of first bank of last record
JW(4)	1	print flag for card image records
JW(5)	5	card input unit
JW(6)	6	print output unit
JW(7)	100	number of messages to be printed
JW(8)	100	number of banks to be printed
JW(9)	4Hbbbb	name (2*A4) of array, defined by default for
JW(10)	4Hbbbb	the arrays in commons /BCS/ and /HCS/
JW(11)		number of array (1...10)
JW(12)		length of array
JW(13)		address of array
JW(14)		first index of named bank area
JW(15)		first index of gap
JW(16)		first index of work bank area
JW(17)		number of deleted words of named banks
JW(18)		number of deleted words of work banks
JW(19)		index of lowest deleted named bank
JW(20)		index of highest deleted work bank
JW(21)		index of link bank
JW(22)–JW(27)		indices of work banks for lists C, E, R, S, T
JW(28)		
JW(29)		index of I/O-statistic bank
JW(30)		check word (=12345)
JW(31)–JW(36)		statistic: number of return codes 1 - 6
JW(37)		
JW(38)		starting at 1, this word is increased by 1 for every garbage collection for named banks or drop of named banks
JW(39)		number of garbage collections (BGARB)
JW(40)		number of garbage collections (WGARB)
JW(41)–JW(50)		histogram of used space before garbage collection in BGARB and WGARB, and in BDROP before the banks are dropped, in 10 percent bins

Table 5: Systems data

type	argument	explanation
ch	DSN	data set name
ch	FMT	format of named bank
		of the type 'f1,f2...........', where f1, f2 ... are format codes nI for integer fields,
		nf for floating point fields , nA for text fields or n(...) for a group format specification,
		For packed 32bit, 16 bit or 8 bit words FMT is 'B32', 'B16' or 'B08'.
	ID	index of a work bank
	IND	index of a named bank
	IW/JW	BOS-array
ch	LIST	list, defining set of banks
		either in the form 'NAM1NAM2...NAMn',
		or a single letter, which may be either C, E, R, S ,T or 0 (empty list).
	LUN	logical I/O-unit number
	IRECL	number of words in a direct access record
	N	sequence number of a name in a list
ch*4	NAME	name of a bank
	NBLK	block size in machine words for standard (machine dependent) writing
	NBLK16	block size in 16 bit words for machine independent writing (should be multiple of 180)
	ND	number of data words in a bank
	NDIM	dimension of array
	NR	number of named bank
	NREC	number of records for direct access data set
	NRMAX	largest number of a named bank in BWIND
ch*2	OPT	option selected in BLIST of the type 'L=' or 'L+' or 'L−'
ch*4	PASSWD	Pass word for direct access data sets
	*	alternate return of the form *statementnr

Table 6: Arguments of BOS subprograms

content of JW(1)	explanation	subprogram
=1	additional words not set to zero	NBANK,MBANK,WBANK,...
=2	move bank to gap of free space	NBANK,MBANK
=3	create bank and drop all other banks	
	of the same name	NBANK,MBANK
=1	drop banks after write	BOSWR
=1 ...3	see text	BDABF

Table 7: Extended arguments in JW(1)

content of JW(2)	explanation	subprogram
1	insuff. space to create new bank	NBANK,MBANK,WBANK,...
2	insuff. space to increase length	NBANK,MBANK,WBANK,...
3	bank not found	BKTOW,BKFRW
4	read error (I/O error)	
5	read error (error in record structure)	
6	read error (insufficient space to store record)	
7	end-of-data	

Table 8: Return codes in JW(2)

1. Return code not zero. Up to JW(7) messages are printed.

2. Printout of banks by user calls. Up to JW(8) banks are printed

The system words JW(6) ...JW(8) can be changed by the user at any time after initialization of the arrays. All printout is suppressed by defining JW(6)=0.

9.2 Error conditions with program stop

All messages concerning errors with program stop are printed on the system output unit JW(6). The error is explained in the printout, for some errors additional information is printed.

1. Initialization errors (in subprogram BOS):
   - basic array IW not first array
   - more than 10 arrays used
   - array to short for initialization

2. Argument errors – wrong argument is printed:
   - array in argument not initialized
   - negative number of data words (ND<0)   (NBANK, MBANK, WBANK)
   - error in argument LIST or OPT   (BLIST)
   - I/O unit used with incompatible operations (read after write etc.)
   - error in format FMT (BKFMT) (for this error the program will stop at the first application of a format)

3. Internal space problem. For certain operations the system needs some space in an BOS-array. If not enough space is reserved for this, the program has to stop. The name of the internal subroutine is printed together with the message: CALL BNRES before initialization to increase space reserved for internal operations by 100 words, in addition a list of existing banks is printed.

4. Too many different names used. If more than the maximum number of different names is used, the program has to stop. A comment is printed, including the message: CALL BNMAX(NMAX) before initialization for a maximum number of NMAX names. The default value for NMAX is 100. In addition a list of all used names is printed. Often this condition will be caused by errors in the user program with the specification of values of the argument name, which really are not names.

5. Bank structure destroyed. In some system subprograms the validity of the bank structure can be checked. If errors are detected, a part of the array is printed before the program stop.

In case of an error with program stop the BOS subprogram BABEND is called. The standard version of this subprogram calls in turn subroutine ABEND, which usually prints a trace back of subprograms before the stop. At a given installation it may be necessary to use a modified version of subprogram BABEND, if no acceptable version of subprogram ABEND is available.

10 Program organisation using BOS

10.1 Modular program structure

The BOS system is primarily designed for the use in the reconstruction and analysis programs of high energy physics experiments. In these programs usually records or events are processed sequentially and a MAIN program should have a modular structure as shown in the figure below.

```
           Initialization
                 V
+------->--------+
|                V
|        +----------------+
|        |                |
|        |  IO - MODULE   |
|        |                |
|        +----------------+
|                V
|                +--------->----------+
|                |                    |
|        +----------------+           V
|        |                |          STOP
|        |   MODULE  1    |
|        |                |
|        +----------------+
|                V
|                .
|                .
|                |
|        +----------------+
|        |                |
|        |   MODULE  n    |
|        |                |
|        +----------------+
|                V
+-------<--------+
```

The processing loop starts with a call to an IO-module, which performs the read and write operations. Then the different processing modules are called.

A module for data processing can be steered completely by data in banks. A complete program for the sequential processing of data records may consist out of a MAIN program, which calls an input/output module (for example BSEQR) and all the processing modules. A new processing module can be added by adding one line in the MAIN program.

A single processing program module can be defined as a set of subprograms with a steering subroutine, which performs a certain task of data reduction (event reconstruction, event analysis). The concept of a program module requires to have a minimum dependence on other programs, not belonging to the module. It is recommended to use the following general conventions:

- all banks are stored in the basic array IW/RW
- all bank names of the current data (event data) are stored in the list 'E'

It is further recommended to observe the following rule in the design of data banks and processing modules: *"A bank with a given name is created and completely filled in one module only, and no changes to the data are made in modules applied later."*

A module requires input data, preferentially banks with a single name, and will produce output data (results), which preferentially are stored in output banks with a single name. The data flow within a module is shown in the figure below.

```
                        +-----+     +-----+     +-----+
   input banks          |     |     |     |     |     |
                        +-----+     +-----+     +-----+
                            \          |          /
                              \        |        /
                                \      |      /
                                  V    V    V
                              +-----------------+
                              |                 |
                              |     MODULE      |
                              |                 |
                              +-----------------+
                                  V    V    V
                                /      |      \
                              /        |        \
                            /          |          \
                        +-----+     +-----+     +-----+
   output banks         |     |     |     |     |     |
                        +-----+     +-----+     +-----+
```

In the example below it is assumed, that the module requires input data in banks with the name BKIN, and produces new data, which are stored in the bank BKUT. A data reduction module could have a structure as shown in the listing below.

```
      SUBROUTINE MODUL
      COMMON/BCS/IW(1000)
      REAL RW(1000)
      EQUIVALENCE (IW(1),RW(1))
      INTEGER NAMI/0/
*     at first entry determine name-indices and define format
*     for output bank
      IF(NAMI.EQ.0) THEN
         NAMI=NAMIND('BKIN')
         NAMU=NAMIND('BKUT')
         CALL BKFMT('BKUT','3I,2F')
      END IF
*     immediate return, if either output bank already exists,
*     or input bank does not exist
      IF(IW(NAMU).NE.0.OR.IW(NAMI).EQ.0) GOTO 100
*     add name of output bank to list 'E'
      CALL BLIST(IW,'E+','BKUT')
*     prepare loop on all input banks
      INDI=NAMI+1
   10 INDI=IW(INDI-1)
      IF(INDI.EQ.0)  GOTO 100
      NR=IW(INDI-2)
*     create output bank with same number as input bank
      INDU=NBANK('BKUT',NR,10)
      IF(INDU.EQ.0) GOTO 100
*     data reduction for data in bank (BKIN,NR), index INDI
*     result will be stored in output bank (BKUT,NR), index INDU
*     ...
      GOTO 10
  100 RETURN
      END
```

In the above program the name-indices for input and output banks are determined at the first entry to allow fast access to the banks. In addition the format of the output bank is defined. Then it is checked, wether computation is necessary. If either the output banks already exist, or the input banks do not exist, computation is not necessary or not possible and the module returns immediately to the calling program. If the user wants to repeat the computation (for example, if the module has been improved), he can drop the output banks before the call of the module by the statement

```
      CALL BDROP('IW,'BKUT')
```

and then the module will create new banks. If computation has to be done in the module, the name of the output banks will be added to the list 'E', in order to get the banks written together with the other data banks. Then a loop is executed over all input banks. Note that the banks numbers do not have to be consecutive.

Options in BSEQR

Subprogram BSEQR may be used directly as the IO-module or may serve as a model for a more specialized version. In high energy physics experiments one record usually contains the data of one event (interaction of elementary particles), which is characterised by a run and an event number. Subroutine BSEQR contains options to select certain events or runs or to ignore certain runs. A fast record skipping method is used. A necessary condition for the use of the option is the storage of the run and event number in the first (header) bank, with a fixed name and number, of all records (note that in writing records the order of banks in a record is defined by the list of names, not by the order of bank creation). If a records does not contain the specified bank as first bank, the record will be accepted.
The user can define the relevant parameters for the bank containing run and event number by a call of BSEQH.

```
CALL BSEQH(NAME,NR,IRUN,IEVT)..........define header bank
```

The meaning of the arguments is: the bank (NAME,NR) will contain the run number at word IRUN and the event number at word IEVT. The default assumption is equivalent to the call

```
      CALL BSEQH('HEAD',0,2,3)
```

Possible options are

- select certain event numbers (SEVT)
- select certain run numbers (SRUN)
- ignore certain run numbers (IRUN).

The priority of the options is in the order given; if for example the option SEVT is used, the options SRUN and IRUN cannot be used. The use of the options require banks with names SEVT, SRUN and IRUN, which most easily are defined on card image records and read with the subroutine BREADC (see next chapter). The content of the card image records is as follows:

```
SEVT  nrun   nevt   nevt   nevt ...         ! selected events
SRUN  nrun   nrun   nrun1 -nrun2 ...        ! selected runs
IRUN  nrun   nrun   nrun   nrun1 -nrun2 ... ! runs to be ignored
```

Any number of these card image records is allowed. The SEVT card image record has to contain the run number as first integer, followed by the event numbers. The SRUN and IRUN card image records contain the run numbers. If a whole range NRUN1...NRUN2 of run numbers is required, this can be specified in the form NRUN1 -NRUN2 (upper limit of range given as a negative number).
If the user wants to skip the run of the current event he may call BSEQS.

```
CALL BSEQS............................skip current run
```

An example of a MAIN program, where all options are used, is given below.

```
      COMMON/BCS/IW(5000)
      CALL BOS(IW,5000)
      CALL BREADC                  read data cards
      CALL BSEQH('HEAD',0,1,2)     define header bank
      CALL BSEQP(5,1000)           define limit parameters
```

```
      LUNR = 1
   10 CALL BSEQR(LUNR,IRET)
      IF(IRET.LT.0) GOTO 100
*     ...
      CALL MODULE
      CALL BSEQW(3)
      IF(...) CALL BSEQW(4)
      IF(...) CALL BSEQW(-3)
*
      IF(...) CALL BSEQS              skip run
      IF(...) CALL BSEQE              force end-condition
*     ...
      GOTO 10
  100 STOP
      END

                                      card image records
SRUN  22345   21234  -21260   21261         ! single beam runs
SRUN  30100   30200   30300  -30399         ! cosmic runs
```

10.2 Module steering

A module should contain the logic for the initialization of the module and other special situations (run start or run end for example). The situation can be determined from the data in banks and is supported by subprogram BSEQR, if the header bank containing run and event number is defined. BSEQR provides some data for the steering of a module in a common /BOSMDL/, explained at the end of this section. Further steering information is provided by the use of the subprogram SMODUL.

SMODUL has to be called as the first executable statement in each module with the module name (called 'SUBNAM' here) as argument:

```
      SUBPROGAM SUBNAM
      COMMON/BCS/  ...
      CALL SMODUL('SUBNAM')
      ...
  100 RETURN
      END
```

The subprogram SMODUL has three functions: book keeping on modules, steering of histogram calls and steering of debug printout.

Book keeping. At a call of SMODUL a bank (BOOK,0) is either created or extended; the name of the module is recorded in two consecutive words (A format) and the date is recorded in a third word (integer). If the name of the module is already contained in the bank BOOK, the date of the previous application of the module is returned (IDATEL) in the steering common and the date within the bank BOOK is overwritten by the actual date.

Histogram steering. Usually a module contains some calls to a histogramming package, to allow checks on the data processing. Histogram packages usually allow only one identifier ID (integer) for each histogram. If several modules include histogram calls, some book keeping is necessary to avoid the multiple use of histogram identifiers in different modules. This book keeping is done by SMODUL and allows up to 99 internal identifiers within one module, independent of identifiers used in other modules. SMODUL returns in the steering common an offset IHA (in steps of 100 for different modules) for each argument 'SUBNAM'. Histogramming calls should be done with identifier ID = IHA + I, where I (internal identifier) is an integer between 1 and 99. If IHA has a negative value, histogrammming calls should be suppressed.

If histogramming in a module should be suppressed, the user of a module has to call

```
      CALL HISTOF('SUBNAM')
```

before the first call of the module SUBNAM. Then in later calls of the module the histogram offset IHA will have the value -1.

The BOS system includes simple histogramming routines, which automatically use an offset for the histogram identifier and allow to suppress the histogramming by a call of HISTOF.

Steering of debug printout. Usually a module contains some debug printout statements. This debug printout should be controlled by the flag IDB in the steering common, with the following meaning:

value	meaning
IDB = 0	no printout
IDB = 1	minimum printout
IDB = 2	more printout
IDB = 3	even more printout

If a user of a module wants debug printout for the following ICOUNT calls of the module with a value IDEB, he has to call

```
      CALL SETDB('SUBNAM',IDEB,ICOUNT)
```

Then in the following ICOUNT calls of the module SUBNAM the debug flag IDB is set to the argument value IDEB, and to zero for the following calls.

The content of the steering common/BOSMDL/ is as follows:

```
*     STEERING COMMON
      LOGICAL       BEGJOB,ENDRUN,BEGRUN,REVENT,ENDJOB,OTHDAT
      COMMON/BOSMDL/BEGJOB,ENDRUN,BEGRUN,REVENT,ENDJOB,OTHDAT,
     +              LCCRUN,NCCRUN,NEVENT,
     +              IHA,IBS,IDB,IDATEL

Logical flags, set at the return from BSEQR (require BSEQH):
         true ...          (false otherwise)
BEGJOB        at first record only
ENDRUN        at the first event after a run (last run is = LCCRUN)
BEGRUN        at first event of a run (run is NCCRUN)
REVENT        if header bank found (event read in)
ENDJOB        after end-of-data (no record read in)
OTHDAT        if no header bank found (no event)

Numerical data, set at the return from BSEQR:

LCCRUN        run number of previous event
NCCRUN        current run number
NEVENT        current event number

Numerical data, set at the return from SMODUL:

IHA           offset for histograms (multiple of 100)
IBS           offset for BOS-histograms
IDB           debug flag (0 means no debug printout)
IDATEL        date, if module already applied to event (else 0)
```

A BOS records

The structure of BOS records, as written by BWRITE and as accepted by BREAD, is described. The body of a record is a sequence of banks, where each bank has (as in storage) a header of four words(name, number of bank, dummy, number of data words ND) followed by ND data words. Reading is faster, if the banks of each name are grouped together, and if the banks of each name are ordered according to increasing number (this is done by BWRITE). The splitting of banks in two or more parts is allowed. Splitting in two parts means, that a certain bank with total ND data words is splitted into two consecutive banks, each with a header of four words as above, with ND1 and ND2 data words (ND=ND1+ND2). In both IO-modes BOS-records are divided into segments, each with a segment code. The segment code IC has the following possible values:

segment code	content
IC = 0	complete BOS-record
IC = 1	first part of a BOS-record
IC = 2	part (neither first nor last) of a BOS-record
IC = 3	last part of a BOS-record

In the mode 'FORT' the body of a BOS-record is splitted into segments, fitting into a given buffer. Each record can be read with an IO-list

```
NTOT, (IBUF(I),I=1,NTOT)
```

The array IBUF will then contain segments. Each segment starts with the words

```
'HIDD', 0, dummy, NT, IC,
```

followed by (NT-1) words, where the (NT-1) words contain banks, and IC is the segment code. In the mode 'EPIO' the standard EPIO physical and logical headers are used except that the logical record type (word 2 of the logical record header) is set to the segment code IC.

B Histograms

The BOS system includes some simple histogramming routines. They use the common/HCS/ and the user has to initialize this common, if one of the calls is used. Each set of programs has, as first argument, the identifier ID. There are two possible ranges of allowed values. If the subprogram SMODUL (see chapter 10) is not used, the identifier ID may be an integer between 1 and 999999999. If however the subprogram SMODUL is used, its argument ID may only be in the range 1 to 99 and the identifier is constructed from the numerical identifier ID (argument) and the name of the subprogram.

One- and two-dimensional histograms

The HIST and CORR subprograms for unweighted one- and two-dimensional histograms do not require booking and automatically determine the bin limits. The bin definition however is possible. One-dimensional histograms have 120 bins. Two-dimensional histograms have 50*100 bins, the printout shows the content of bins with an 'X' for more than 6 counts in a bin; the projections are printed in addition.

```
1-dimensional histogram, 120 bins
CALL DHIST(ID,XA,XB)..............define limits for 1-dim histogram
CALL THIST(ID,TEXT)...............define text
CALL UHIST(ID,X)..................entry for value x
CALL PHIST(ID)....................print histogram (ID=0 -> all)

2-dimensional histogram, 50*100 bins
CALL DCORR(ID,XA,XB,YA,YB)........define limits for 2-dim histogram
CALL TCORR(ID,TEXT)...............define text
CALL UCORR(ID,X,Y)................entry for pair x,y
CALL PCORR(ID)....................print histogram (ID=0 -> all)
```

C Summary of BOS

```
            (Copy of the DESY interactive HELP)

            OOOOO     OOOO     OOOO
            O    O   O    O   O    O
            O    O   O    O   O
  HELP for  OOOOO    O    O    OOOO     FORTRAN 77 version
            O    O   O    O        O
            O    O   O    O   O    O
            OOOOO     OOOO     OOOO

Please select your field of interest with a * in col.1:
select     Function calls
select     Subroutine calls
select     Arguments
select     Data cards
select     General IO program
select     Model job
select     BOS histogramming programs
select     Libraries at DESY
select     Export of BOS source code
select     Modifications

Send comments to Volker Blobel (F14BLO at DHHDESY3)
           or to Otto Hell     (R01HLL at DHHDESY3)
           or to Dieter Lueke  (F35LUE at DHHDESY3)
                                            (use ((SENDMAIL)) at DESY)

                                                                   page
                                                             numbers are
                                                              for manual
-------------------------------------------------------------------page
Function calls:

CHST = CHAINT(INST) ...... *4 ..conversion to character string ....  5
INST = INTCHA(CHST) ..............conversion from character string   5
IND  = MBANK(JW,NAME,NR,ND) ......create named bank ...............  7
IND  = MDARD(JW,LUN,NAME,NR)......read bank from da-ds............. 23
IND  = MDARS(JW,LUN) .............read da-ds sequentially.......... 23
IND  = MDROP(JW,NAME,NR) .........drop named bank..................  7
IND  = MLINK(JW,NAME,NR) .........find index of named bank.........  7
IND  = MPRNT(JW,NAME,NR) .........print named bank.................  7
IND  = MREADC(JW) ................read bank from cards............. 19
IND  = MSWAP(JW,NAM1,NR1,NAM2,NR2)exchange name and number......... 12
NAMI = NAMIND(NAME) ..............get name-index fom name..........  8
IND  = NBANK(NAME,NR,ND) .........create named bank................  7
NNR  = NDANR(LUN,NAME,LEG,NR).....get number of bank on da-ds...... 22
IND  = NDROP(NAME,NR) ............drop named bank..................  7
IND  = NLINK(NAME,NR) ............find index of named bank.........  6
NAME = NLIST(JW,N,LIST) .. *4 ..get name from list................ 11
IND  = NPRNT(NAME,NR) ............print named bank.................  7
IND  = NSWAP(NAM1,NR1,NAM2,NR2) ..exchange name and number......... 12
                             *4 means character*4 function
```

```
-----------------------------------------------------------------page
Subroutine calls:
```



```
Subroutine calls for printout of general information:
```



```
---------------------------------------------------------------------
Arguments:

DSN ..................... * .......data set name
FMT ..................... * .......format of named bank
ID ................................index of a work bank (in a common)
IND ...............................index of a named bank
IW/JW .............................BOS-array
LEG ..................... *2 ......'LE' or 'EQ' or 'GE'
LIST .................... * .......list, defining set of banks
LUN ...............................logical I/O-unit number
IOMODE................... * .......I/O mode, 'FORT' or 'EPIO'
IRECL .............................nr of words in a da record
N .................................sequ. nr of a name in a list
NAME .................... *4 ......name of a bank
```

```
NBLK ...............................buffer size for IO
ND  ................................number of data words in a bank
NDIM ...............................dimension of BOS-array
NR .................................number of named bank
NREC ...............................number of records for da-ds
NRMAX ..............................largest nr of a named bk (BWIND)
OPT ..................... *2 ......option selected in BLIST
PASSWD .................. * ........password for da ds
*ret................................alternate return
                           * means character string

----------------------------------------------------------------------
Data cards:

name      ...data...  ! comment       |  *  comment
name nr / ...data...                  |  UNIT unit
name $FMT 'bank format'               |  POFF
                                      |  PON
name nr / $TEXT or $TEXTC             |  ENDQ
..data cards                          |
END$                                  |

----------------------------------------------------------------------
General IO program and module control program:

CALL BSEQP(NSEC,NLIM)............parameter for general IO program. 17
CALL BSEQH(NAME,NR,IRUN,IEVT).....specify header bank.............. 30
CALL BSEQR(LUNR,IRET)............general IO program/input of rec.  17
CALL BSEQW(LUNW).................set flag for output.............. 17
CALL BSEQS.......................skip current run................. 30
CALL BSEQE.......................end condition.................... 17

CALL HISTOF('subnam')............suppress histograms.............. 31
CALL SETDB('subnam',IDEB,ICOUNT)..set debug flag.................. 32
CALL SMODUL('subnam')............start module..................... 31

Model job including MAIN program:

//         JOB
//*
//          EXEC JFORTCLG,                      Siemens compiler
//          LLB1='R01UTL.BOS.LF77',             LIBRARIES...
//          LLB2='R01UTL.CERN.PACKLIB'
//*                                                  or
//*
//*         EXEC VFORTCLG,                      IBM compiler
//*         LLB1='R01UTL.BOS.LFVS',             LIBRARIES...
//*         LLB2='R01UTL.CERN.PACKLIB4'
//*
*     ***** Model  M A I N   program *******
*     declare BOS commons and initialize BOS
*     BASIC BOS common for data processing
      PARAMETER (LENBCS=50000,LENHCS=10000)
      COMMON/BCS/IW(LENBCS)
      REAL       RW(LENBCS)
      EQUIVALENCE(RW(1),IW(1))
*     Common /HCS/ for BOS histograms
```

```
      COMMON/HCS/KW(LENHCS)
      CALL   BOS(IW,LENBCS)
      CALL   BOS(KW,LENHCS)
*     read data cards
      CALL BREADC
*     define parameter: CALL BSEQP(NSEC,NLIM)
*     NSEC = nr of seconds> stop NSEC seconds before reaching timelimit
*     NLIM = nr of records> stop after NLIM records. 0 = infinity
      CALL BSEQP(2,0)
*     run and event number in 'HEAD',0 at words 2 and 3
      CALL BSEQH('HEAD',0,2,3)
*     select IO mode, if necessary
C     CALL BUNIT(UNIT,IOMODE,NBLK)
      LUNR=1
*     event processing loop
   10 CALL BSEQR(LUNR,IRET)
      IF(IRET.LT.0) GOTO 100
*     call modules here
      CALL SUBNAM
      GOTO 10
  100 STOP
      END
      SUBROUTINE SUBNAM
*     Standard BOS common
      COMMON  /BCS/IW(1000)
      REAL         RW(1000)
      EQUIVALENCE (RW(1),IW(1))
*     Common with steering data:
      LOGICAL       BEGJOB,ENDRUN,BEGRUN,REVENT,ENDJOB,OTHDAT
      COMMON/BOSMDL/BEGJOB,ENDRUN,BEGRUN,REVENT,ENDJOB,OTHDAT,
     +              LCCRUN,NCCRUN,NEVENT,
     +              IHA,IBS,IDB,IDATEL
*     comment to common above
*     Logical flags, set at the return from BSEQR
*             true ...          (false otherwise)
*     BEGJOB        at first record only
*     ENDRUN        at the first event after a run (last run is= LCCRUN)
*     BEGRUN        at first event of a run (run is NCCRUN)
*     REVENT        if header bank found (event read in)
*     ENDJOB        after end-of-data (no record read in)
*     OTHDAT        if no header bank found (no event)
*
*     Numerical data, set at the return from BSEQR:
*
*     LCCRUN        run number of previous event
*     NCCRUN        current run number
*     NEVENT        current event number
*
*     Numerical data, set at the return from SMODUL:
*
*     IHA           offset for histograms (multiple of 100)
*     IBS           offset for BOS-histograms
*     IDB           debug flag (0 means no debug printout)
*     IDATEL        date, if module already applied to event (else 0)
*     ...
*     module steering by SMODUL
      CALL SMODUL('SUBNAM')
```

```
*
      IF(BEGJOB) THEN
*        at first record only
      END IF
      IF(ENDRUN) THEN
*        at the first event after a run (last run is= LCCRUN)
      END IF
      IF(BEGRUN) THEN
*        at first event of a run (run is NCCRUN)
      END IF
      IF(REVENT) THEN
*        header bank found (event read in) - process data
      END IF
      IF(ENDJOB) THEN
*        after end-of-data (no record read in)
      END IF
  100 RETURN
      END
//G.FT01F001 DD DUMMY
//G.SYSIN DD *
.. (data cards)

Data cards for run and event selection (requires BSEQH):

SEVT  nrun   nevt   nevt   nevt  ...   ! selected events
SRUN  nrun   nrun   nrun1 -nrun2 ...   ! selected runs
IRUN  nrun   nrun1 -nrun2  nrun  ...   ! runs to be ignored


----------------------------------------------------------------------
BOS histogramming programs

Initialization of COMMON/HCS/KW(ndim) required.
Histogram identifier ID:
  ID = 1...9999999, if SMODUL not used
  ID = 1........99 with SMODUL (same ID in different modules allowed)
     each call (including print-call) refers to selected module only

HIST and CORR subprograms for 1- and 2-dimensional, unweighted histo-
grams, automatic bin size determination, if limits undefined (booking
of histograms not necessary)

1-dimensional histogram, 120 bins
CALL DHIST(ID,XA,XB)............define limits for 1-dim histogram.....
CALL THIST(ID,TEXT)............define text..........................
CALL UHIST(ID,X)...............entry for value x....................
CALL PHIST(ID).................print histogram (ID=0 -> all).........

2-dimensional histogram, 50*100 bins
CALL DCORR(ID,XA,XB,YA,YB)......define limits for 2-dim histogram.....
CALL TCORR(ID,TEXT)............define text..........................
CALL UCORR(ID,X,Y).............entry for pair x,y...................
CALL PCORR(ID).................print histogram (ID=0 -> all).........

----------------------------------------------------------------------
Libraries at DESY:
```

```
For the IBM MVS system at the DESY computer center the load libraries
for the  BOS system  and for the  CERN  PACKLIB, required by the  BOS
system, depend on the flavour of FORTRAN 77 that you use:
Recommended: IBM VS FORTRAN (FVS):
      LLB1=R01UTL.BOS.LFVS
      LLB2=R01UTL.CERN.PACKLIB4    FORTRAN IV version, FVS not yet av.
Supported: Siemens FORTRAN 77 (F77):
      LLB1=R01UTL.BOS.LF77
      LLB2=R01UTL.CERN.PACKLIB

The procedures VFORT.. should be used to get the IBM compiler,
the procedures JFORT.. should be used to get the Siemens compiler, see
      Help PROCS

----------------------------------------------------------------------
Export of BOS source code:

The BOS system is available in (only sligthly) different versions  for
the following types of computer: IBM, VAX, GOULD, APOLLO (CRAY version
not yet ready)

Conditions for use of BOS outside DESY:
Organizations  collaborating  with  DESY,  Physics  departments  in
Universities and other non-profit making bodies may receive the source
without charge (contact person: Otto Hell - R01).
Commercial enterprises should ask
  H. Pingel - DESY Technologie Transfer
for conditions.

The programs are provided solely for the use of  the  organization  to
which they are distributed, and may not be redistributed or reproduced
in large numbers without the express agreement of DESY.

----------------------------------------------------------------------
Modifications:

                                                          (December 87)
     Direct access reading will be  four  times  faster  in  the  next
     version (at least with the Siemens compiler)
                                                          (November 87)
     Some known bugs fixed
     New libraries (for SIEMENS and IBM Fortran77) AT DESY
     Common name /CMODUL/ changed to /BOSMDL/

 !   WARNING: pointer to work banks should be in a labelled common;
     if not, an optimizing compiler may produce wrong code.

     CHANGES in BDABF and INITDA: OPEN for direct access data sets

The OPEN statement in various compilers and machines shows,  how  many
good ideas compiler-writers have and how bad the standard OPEN is. The
consequence  is,  that  many  machine  dependent   extensions   exist,
sometimes with unknown properties. This makes it difficult  to  use  a
common OPEN-statement. In the previous BOS version and in the  present
version there are OPEN statements for direct access data sets in
    SUBROUTINE INITDA   initialization
```

```
   SUBROUTINE BDABF    open direct access unit for BOS

Now the user may execute his private OPEN statement and can instruct
BOS not to use the internal OPEN statement.

Initialization:
   CALL INITDA( ...      if BOS OPEN statement should be used

   CALL INITDU( ...      if the user has his own statement
                identical arguments

Opening for BOS (BDABF):

   set additional argument in IW(1) before the call of BDABF
   define data set name in argument of BDABF call, at least for
   IW(1)=0 or 1.

IW(1)=0 (default)
        BOS OPEN has status = 'UNKNOWN'
        Use DISP=SHR in DD-statement at DESY IBM

IW(1)=1 exclusiv access assumed (faster, but only one user at a time)
        BOS OPEN has status = 'OLD'

IW(1)=2 User OPEN-statement (no internal BOS OPEN executed)
        shared access
        Use DISP=SHR in DD-statement at DESY IBM

IW(1)=3 User OPEN-statement (no internal BOS OPEN executed)
        exclusiv access assumed

followed by CALL BDABF(LUN,IRECL,DSN,PASSWD)
```