ALEPH 86 – 143
S.J. Wheeler
December 6, 1986
Distr: DATACQ/SOFTWR

A Database Based on BOS

Abstract

This document describes a number of subroutines which have been written to enable tasks to store and retrieve banks of constants from a direct access disk file. The package is written in Fortran77 and uses the BOS memory management system. The database package is available on all the ALEPH VAXes at CERN.

A Database Based on BOS

Author:	S.J.Wheeler
Version of Document:	1.0
Revision date:	29 November 1986
Status:	Version 1.0

Contents

Contents contd.

1. Introduction

This document describes a number of subroutines which have been written to enable tasks to store and retrieve banks of constants from a direct access disk file. The package is written in Fortran77 and uses the BOS memory management system (ref. 1) .

In the data acquisition system, it is necessary to have some form of temporary storage system for sets of calibration constants produced by monitoring tasks before they are written to tape. As the constants have to be written to tape in the form of BOS banks it was decided that the direct access I/O facility of the BOS system would be the most convenient method for storing the information. A description of this system may be found in Chapter 7 of reference 1. In addition, it is also useful to have a means of storing miscellaneous data, for example run description information. By using the BOS direct access routines a subroutine library has been written which will maintain a database of BOS banks on a direct access file. The subroutines provide an interface between the user and the database.

It was decided that the following main functions were required of the interface, namely the ability to write banks to the database, read banks from the database, remove banks from the database, produce an index of banks on the database and write banks from the database to tape. In addition to this, utility routines are required to initialise the database and perform a garbage collection when necessary. Finally, the ability to maintain several different databases was thought to be a desirable feature.

One problem that had to be resolved was how to allow simultaneous access to the dataset by several different tasks at a time without :-

a) two jobs attempting to access the same record on the direct access dataset at the same time
b) a task trying to read a set of banks from the database when a garbage collection is taking place.

BOS provides a mechanism to prevent case a) whereby the dataset is reserved temporarily for each task that wishes to access the dataset using the system records of the dataset. However, this method is slow and can also be unreliable, there is no obvious method in BOS to solve case b). Therefore, it was decided to use the VAX Lock Management Facility of the VMS System Services (reference 2) to reserve access to the dataset, which not only removes the danger of clashes between processes, but also speeds up the reservation time considerably. The same system is used to prevent tasks from accessing the database when a garbage collection is in progress.

2. Implementation Details and Database Use

The database has already been installed on all the ALEPH VAXes (VXALFB, VXALBM, VXALTP and VXALZO). However, the following notes will be useful for anyone wishing to use the database or install the package on other VAX machines.

2.1 File Organisation

On each VAX the following directories have been created with the corresponding logical names.

Physical Name	Logical Name	Contents
[ONLINE.DATABASE.SOURCE]	A_DBA$SRC	Contains all the fortran source files and include files for the database package
[ONLINE.DATABASE.NODEB]	A_DBA$DIR	Contains all the object files and the object library for the database
[ONLINE.DATABASE.MGR]	A_DBA$MGR	Contains the help library for the database package

2.2 Naming Conventions

The following convention has been adopted for the routine names used in the package

DBA_Action_Object

where DBA serves as a label to identify the routine as coming from the DataBAse package. The second part of the name indicates the action being performed by the routine and the final part is the object on which the action is performed e.g. DBA_DEFINE_BANK. In some routines the object is divided into 2 parts e.g. DBA_LIST_ALL_BANKS. All the database routines, may be found in the object library :–

A_DBA$DIR:NEWDBA.OLB

2.3 Include Files

All the routines described in this report may be called as INTEGER FUNCTIONS where the value of the function is the status value returned from the routine. The possible status values are defined in a parameter file which may be included in user routines as follows:-

```
INCLUDE    'A_DBA$SRC:DBASSDEF.INC'
```

A list of all status codes included in this parameter file is given in Appendix A. The codes may be used with the VAX/VMS Run Time Library routine LIB$SIGNAL. The names of the routines themselves may be included in user routines as follows :-

```
INCLUDE    'A_DBA$SRC:DATABASE.INC'
```

2.4 Creating the Database

In order to allow a variable number of databases, each database has to be assigned a different logical unit number by which it can be identified and refered to. By default the name of the database file will be FOR0XX where XX is the logical unit number. If the user wishes the file to have an alternative name it is necessary to define it using the DCL command ASSIGN. For example :–

```
ASSIGN    DISK$USER:[USERID.DATABASE]MYDAT.DAT    FOR040
```

associates the dataset name DISK$USER:[USERID.DATABASE]MYDATA.DAT to logical unit number 40. If the name assignment is to be system wide (that is if users with different UICS are going to access the database) then the ASSIGN command has to be used with the SYSTEM option and the user has to have the corresponding privilege to assign the logical name.The ALEPH DAQ database has been assigned to logical unit 92 on the VXALFB, VXALBM, VXALTP and VXALZO VAXes, the name of the database is:–

```
DISK$USER:[ONLINE.DATABASE]DATABASE.DAT
```

Once a name has been assigned the dataset may be created by running a job incorporating the routine DBA_INIT_DATABASE described in section 3.1.1. An example of such a job may be found in:–

```
A_DBA$SRC:CREATE_DATABASE.FOR
```

2.5 Using the Database

Once created the database may be used freely in the following manner. The user must define in their program a BOS array 20000 words long with the following FORTRAN statement:–

```
COMMON /BCS/ IW (20000)
```

This array is used by the database package. The first call to the package must be the database initialisation routine DBA_INIT_PACKAGE descibed in section 3.3.2. Once this has been done, any of the other database routines may be called.

A simple example of how to use the database package routines is given in Appendix B. Another example may be found in:–

```
A_DBA$SRC:INQUIRE_DB.FOR
```

This uses the UPI user interface package to present the user with a menu from which they can choose to LIST all the banks on the database, DUMP a bank from the database to the terminal screen, INCLUDE a bank in the database, REMOVE a bank from the database or REPLACE a bank on the database.The corresponding executable image may be found in:–

```
A_DBA$DIR:INQUIRE_DB.EXE
```

2.6 Linking Programs with the Database Package

As mentioned earlier in section 2.2 all the database routines are contained in the object library A_DBA$DIR:NEWDBA.OLB. Since all the database routines use BOS it is necessary to link this in addition. On VXALBM, VXALTP and VXALZO the BOS library may be found in [ONLINE.BOS]JHBOS77.OLB. A link to 'CERN$LIBS' is also required.

On VXALFB the BOS library has now been installed as a sharable image. In order to link it 2 option files are required:–

```
[ONLINE.BOS]BOS.OPT and
[ONLINE.BOS]BOS_COMMON.OPT
```

A link to 'CERN$LIBS' is not required.

2.7 Privileges Needed

As mentioned earlier the VAX Lock system service is used to reserve access to the database. Since the reservation in general has to be system wide, it is necessary for all processes using the database package to have the **SYSLCK** privilege.

2.8 Limitations

There is no limit on the size of banks which can be saved on the database, or on the number of banks which may be saved (the database is automatically extended if it becomes full). There is no limit on the bank numbers of the banks saved on the database, provided that the bank number is zero or a positive integer, negative bank numbers are used by the database package itself.

2.9 Help !

A help library has been written which contains shortened versions of all the database routine descriptions given in Section 3. It has been installed on each VAX and may be accessed by typing HELP DBA. The help library itself may be found in:–

A_DBA$MGR:DBA.HLB

3. The Routines

All the routines used in the database package are described in this section.

3.1 DBA_INIT_DATABASE.

DBA_INIT_DATABASE is used to ceate a new database.

FORMAT	**DBA_INIT_DATABASE**	lun_db
RETURNS	type:	**longword(unsigned)**
	access:	**write only**
	mechanism:	**by value**
ARGUMENTS	***lun_db***	
	type:	**longword**
	access:	**read only**
	mechanism:	**by reference**
	Logical unit number of database to be created	

DESCRIPTION

If a new database is to be created then it is necessary to create the dataset (after a logical unit has been assigned to it) using the above routine. This routine needs only to be used once. It should *not be used* after the banks have been added to the database since its effect is to erase everything on the dataset. It is recommended that this routine is used in a separate job only to be run before the database is used.The routine creates a dataset with 150 records, each with length 1000 words.

N.B. This routine should be used with **caution !**

3.2 DBA_INIT_PACKAGE.

DBA_INIT_PACKAGE is used to open an existing database.

FORMAT **DBA_INIT_PACKAGE** lun_db, lun_output

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database to be opened

lun_output

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of file to which BOS output messages should be sent

DESCRIPTION

This routine initialises the BOS array which must have been defined in the calling routine to be 20000 words long. The database on logical unit number **lun_db** is opened. The output destination to which all BOS output messages are sent is defined e.g. **lun_output** = 6 will result in all messages being sent to the screen. BOS messages can be supressed by setting **lun_output** to 0. The output destination is defined in the routine by setting word 6 of the BOS array to the desired logical unit number. If, for example, the user wishes to reenable output after they have set **lun_output** to 0 then they have to do it by setting word 6 of the BOS array directly. This routine *must* be called once by each process using the database package before using any of the following routines, otherwise they will all return the error code DBA_SS_NOT_INIT.

N.B. This routine is the same as the routine DBA_INIT described in the last version of this document.

3.3 DBA_DEFINE_NEW_BANK.

DBA_DEFINE_NEW_BANK is suitable for defining a new bank, which does not already exist on the database.

FORMAT	**DBA_DEFINE_NEW_BANK**	lun_db, bank_name, bank_number, bank_length, bank_index

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database on which bank is to be stored eventually

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank to be defined

bank_number

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Number of bank to be defined

bank_length

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Length of bank to be defined

bank_index

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Index of defined bank in calling routine's BOS array

DESCRIPTION

Defines the bank with specified **bank_name**, **bank_number** and **bank_length** in the calling routine's BOS array. The index of the bank is returned as **bank_index**. If the bank already exists on the database then the bank is not defined and an error code is returned as the function value.

3.4 DBA_DEFINE_BANK.

DBA_DEFINE_BANK is suitable for defining a bank which already exists on the database and the user wishes to replace.

FORMAT **DBA_DEFINE_BANK** lun_db, bank_name, input_length, bank_number, output_length, bank_index

RETURNS

type: **longword(unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

lun_db
type: **longword**
access: **read only**
mechanism: **by reference**
Logical unit number of database on which bank is to be stored eventually

bank_name
type: **character string**
access: **read only**
mechanism: **by descriptor**
Name of bank to be defined

input_length
type: **longword**
access: **read only**
mechanism: **by reference**
Desired length of bank to be defined

bank_number
type: **longword**
access: **write only**
mechanism: **by reference**
Number assigned to the defined bank

output_length

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Length assigned to the defined bank

bank_index

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Index of defined bank in the calling routine's BOS array

DESCRIPTION

Defines a bank with the given **bank_name** in the calling program's BOS array. The bank number is defined to be equal to the highest bank number for that name on the disk. If the name does not exist, then **bank_number** is set to 1. If the bank already exists on disk then a check is made to see if **input_length** is the same as the length for the bank on disk. If the lengths are not the same then the bank is defined with the same length as the bank already saved and this is returned in the **output_length** argument. In which case the error code DBA_SS_LENGTH_CHANGED is returned as the function value. The bank index is returned as **bank_index**.

3.5 DBA_INCLUDE_BANK.

DBA_INCLUDE_BANK is suitable for adding banks to the database if the user does not wish to overwrite previous versions of the bank.

FORMAT	**DBA_INCLUDE_BANK**	lun_db, bank_name, bank_number [,bank_format]

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database on which bank is to be stored

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank to be stored

bank_number

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Number of bank to be stored

bank_format

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Format of bank to be stored. If omitted bank is stored with integer format.

DESCRIPTION

This routine stores the specified bank on the database. An optional bank format up to 40 characters long may be specified for the bank. A description of the structure of bank format strings may be found in section 3.3 of reference 1. The routine does not store the bank if an error condition is detected. For example it does not store the bank if it already exists on the dataset (DBA_REPLACE_BANK needs to be used if this function is required). Checks are made to see that the bank has been defined by the calling routine and that the bank number is valid.An automatic garbage collection is called if the database is full. The routine waits until the garbage collection has completed and then saves the bank. The bank is *not* dropped from the calling routine's BOS array once it has been added to the database.

Note: If a bank format has been specified the format is saved by the routine in the following manner. A bank is defined with the same name as the bank to be saved and bank number = – (**bank_number** + 1). The format string is written into the bank and then the bank is saved on the database.

3.6 DBA_REPLACE_BANK.

DBA_REPLACE_BANK is suitable for adding or replacing banks on the database.

FORMAT	**DBA_REPLACE_BANK**	lun_db, bank_name, bank_number [,bank_format]

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database on which bank is to be replaced

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank to be replaced

bank_number

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Number of bank to be replaced

bank_format

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Format of bank to be replaced. If omitted bank is stored with integer format if it does not already exist, otherwise it is stored with the format of the bank it replaces

DESCRIPTION

This routine writes a bank with the given name and number to the database overwriting any bank with *exactly* the same specifications. An optional bank format, up to 40 characters long may be specified. If a bank of the same name and number already exists on the database, but has a different length then that bank is deleted from the dataset and the new bank is added to the last free record of the dataset. If an error condition is detected then the bank is not written to the dataset. The bank is *not* deleted from the calling routine's BOS array once it has been written to the database. An automatic garbage collection is called if the database becomes full.

Note: If the optional bank format is specified the format will be saved in the same way as for DBA_INCLUDE_BANK. This means that if a new format is specified for the bank to be replaced it will overwrite the old format.

3.7 DBA_READ_SINGLE_BANK.

DBA_READ_SINGLE_BANK should be used to retrieve banks from the database.

FORMAT **DBA_READ_SINGLE_BANK** lun_db, bank_name, bank_number, bank_index, flag

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database from which bank is to be read

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank to be read

bank_number

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Number of bank to be read

bank_index

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Index of bank in calling routine's BOS array *flag*

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Flag set to 1 if bank exists on database, set to 0 if it does not

DESCRIPTION

This routine reads the bank of the given name and number from the database into the calling routine's BOS array. It returns the index of the bank as **bank_index**. If a bank format has been defined for the bank this is retrieved and redefined for the bank using a call to the BOS routine BKFMT. The flag is set to 1 or 0 depending on whether the bank exists or does not exist on the dataset.

3.8 DBA_LIST_ALL_BANKS.

DBA_LIST_ALL_BANKS may be used to obtain lists of the names, numbers and lengths of banks stored on the database.

FORMAT	**DBA_LIST_ALL_BANKS**	lun_db, maximum, name_array, number_array, length_array, total

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database on which banks to be listed are stored.

maximum

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Maximum number of banks to be listed

name_array

type:	**character string**
access:	**write only**
mechanism:	**by descriptor, array reference**

Array containing the names of the banks on the database

number_array

type:	**longword**
access:	**write only**
mechanism:	**by reference, array reference**

Array containing the numbers of the banks on the database

length_array

type: **longword**

access: **write only**

mechanism: **by reference, array reference**

Array containing the lengths of the banks on the database

total

type: **longword**

access: **write only**

mechanism: **by reference**

Total number of banks listed

DESCRIPTION

Returns the names (in alphabetical order), numbers and lengths of all the banks on the database along with the total number of banks up to a maximum defined by the **maximum** argument which may be any positive integer. The arrays for the names, numbers and lengths in the the calling routine should be defined to be equal to or larger than **maximum**. If there are more than **maximum** banks on disk the error code DBA_SS_TOO_MANY_BANKS is returned as the function value.

3.9 DBA_LIST_NAMED_BANK.

DBA_LIST_NAMED_BANK may be used to obtain a list of bank numbers and bank lengths for a given bank name saved on the database.

FORMAT	**DBA_LIST_NAMED_BANK**	lun_db, maximum, bank_name, number_array, length_array, total

RETURNS

type: **longword(unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

lun_db
type: **longword**
access: **read only**
mechanism: **by reference**
Logical unit number of database on which banks to be listed are stored

maximum
type: **longword**
access: **read only**
mechanism: **by reference**
Maximum number of banks to be listed

bank_name
type: **character string**
access: **read only**
mechanism: **by descriptor**
Name of bank bank to be listed

number_array
type: **longword**
access: **write only**
mechanism: **by reference, array reference**
Array containing the bank numbers of banks with the same name on the database

length_array

type: **longword**

access: **write only**

mechanism: **by reference, array reference**

Array containing the bank lengths of banks with the same name on the database

total

type: **longword**

access: **write only**

mechanism: **by reference**

Total number of banks with the same name listed

DESCRIPTION

Returns the numbers (in numerical order) and lengths of all the banks on the specified database with the same name up to a maximum defined by the **maximum** argument. If there are more than banks than the specified maximum on the dataset with the same name the error code DBA_SS_TOO_MANY_BANKS is returned as the function value.

3.10 DBA_LIST_SINGLE_BANK.

DBA_LIST_SINGLE_BANK may be used to search the database for a specific bank.

FORMAT **DBA_LIST_SINGLE_BANK** lun_db, bank_name, bank_number, bank_length, flag

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database on which banks to be listed are stored

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank to be searched for

bank_number

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Number of bank to be searched for

bank_length

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Length of bank if found

flag

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Flag set to 1 if bank found on database, set to 0 if bank not found

DESCRIPTION

This routine searches the dataset for the bank with the specified name and number. If found it returns the length of the bank and sets **flag** to 1. If not found **flag** is set to 0.

3.11 DBA_TOTAL_BANKS.

DBA_TOTAL_BANKS may be used to find the total number of banks saved on a database.

FORMAT	**DBA_TOTAL_BANKS**	lun_db, total

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database for which total number of banks is to be found

total

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Total number of banks on the database

DESCRIPTION

This routine returns the total number of banks stored on the specified database. It should be noted that *all* banks are in the total, including banks used by the database package itself (i.e. banks with numbers less than zero).

3.12 DBA_LIST_HIGHEST_BANK.

DBA_LIST_HIGHEST_BANK may be used to find the highest bank number for the given bank name on the database.

FORMAT	**DBA_LIST_HIGHEST_BANK** lun_db, bank_name, bank_number, bank_length

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database to be searched

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank for which highest number is to be found

bank_number

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Highest bank number for given name

bank_length

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Length of bank with highest number

DESCRIPTION

This routine finds the the highest bank number for the given name. In addition to the number it returns the length of the bank with the highest bank number.

3.13 DBA_REMOVE_NAMED_BANK.

DBA_REMOVE_NAMED_BANK may be used to remove a set of banks with the same name from the database.

FORMAT **DBA_REMOVE_NAMED_BANK** lun_db, maximum, bank_name, total

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database from which banks are to be removed

maximum

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Maximum number of banks to be removed

name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of banks to be removed

total

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Total number of banks removed

DESCRIPTION Deletes all banks of the same name from the database up to a maximum defined by **maximum**. If there are more than **maximum** banks of the same name on the database the error code DBA_SS_TOO_MANY_BANKS is returned as the function value.

3.14 DBA_REMOVE_SINGLE_BANK.

DBA_REMOVE_SINGLE_BANK may be used to remove a specified bank from the database.

FORMAT	**DBA_REMOVE_SINGLE_BANK**	lun_db, bank_name, bank_number

RETURNS

type:	**longword(unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lun_db

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Logical unit number of database from which bank is to be removed

bank_name

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Name of bank to be removed

bank_number

type:	**longword**
access:	**read only**
mechanism:	**by reference**

Number of bank to be removed

DESCRIPTION

This routine removes the specified bank from the dataset (if the bank exists). An error code is returned as the function value if the bank does not exist.

3.15 DBA_GARBAGE_COLLECT.

DBA_GARBAGE_COLLECT may be used to garbage collect the database.Since BOS does not reuse the space obtained by deleting banks from the direct access dataset it is necessary to perform occasional garbage collections on the dataset in order to regain the unused space.

FORMAT	**DBA_GARBAGE_COLLECT** lun_db
RETURNS	type: **longword(unsigned)** access: **write only** mechanism: **by value**
ARGUMENTS	***lun_db*** type: **longword** access: **read only** mechanism: **by reference** Logical unit number of database to be garbage collected
DESCRIPTION	This routine performs a garbage collection on the database specified by **lun_db**. All the banks on the dataset are copied to a sequential dataset on logical unit number 91.The dataset is then reinitialised to remove all the old banks and the size increased if it was found to be completely full. The banks are copied back from the sequential dataset onto the database dataset and the sequential dataset is deleted. No other process is able to access the database whilst a garbage collection is taking place. N.B. The garbage collection procedure takes a long time, therefore it should be used as little as possible. For this reason the dataset created on the VAX to hold the database, is large (it is capable of storing approx. 12000 words of BOS bank data). Secondly, it is better to use the DBA_REPLACE_BANK routine if a bank is to be replaced on the dataset rather than to delete the old bank (using DBA_DELETE_SINGLE_BANK) and then add the new one (using DBA_INCLUDE_BANK). The reason for this is that the REPLACE routine overwrites the old record in the database whereas the INCLUDE routine will add the bank to the first unused record, even if there are previous records which are empty because banks have been removed from them.

3.16 DBA_PUT_BANKS_ON_TAPE.

DBA_PUT_BANKS_ON_TAPE does not write banks directly to tape but sends a message, using the UPI message facility (reference 3), to the tape task telling it which banks to read from the database and write to tape.

FORMAT	**DBA_PUT_BANKS_ON_TAPE**	lun_db, number_banks, bank_names, bank_numbers

RETURNS

type: **longword(unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

lun_db
type: **longword**
access: **read only**
mechanism: **by reference**
Logical unit number of database on which banks to be written are stored

number_banks
type: **longword**
access: **read only**
mechanism: **by reference**
Number of banks to be written to tape

bank_names
type: **character string**
access: **read only**
mechanism: **by descriptor, array reference**
Array of bank names to be written to tape

bank_numbers
type: **longword**
access: **read only**
mechanism: **by reference, array reference**
Corresponding array of bank numbers to be written

DESCRIPTION

This routine constructs a message to be sent to the tape task containing a list of the names and numbers of banks the user wishes to be written to tape. The message is in the form of a character string and is equivalenced to an integer message_buffer array. The message_buffer has the following structure:-

MSG_BUFFER(1) = 'DTB'	flag word to indicate this is a message from the database package
MSG_BUFFER(2) = NUMBER_BANKS	total number of banks to be written
MSG_BUFFER(3) = LUN_DB	logical unit number of database
MSG_BUFFER(4) = BANK1_NAME	name of first bank
MSG_BUFFER(5) = BANK1_NUMBER	number of first bank
. .	.
. .	.
MSG_BUFFER(N) = 0	flag to indicate end of list

Before writing the name and number into the list a check is made to see that the data acquisition tape task is active.
Note: This routine is the same as EVIO_PUT_DBA_BANKS_ON_TAPE.

4. References

1. The BOS System – Volker Blobel ALEPH 86-62

2. VAX/VMS Volume 5A System Services

3. User Interface Package – C.Arnault, J.Bourotte, A.Lacourt ALEPH 86-27

5.Appendix A. Completion Codes Returned by the Database Package

A.1 Normal Completion

DBA_SS_NORMAL
Routine has completed successfully

A.2 Error Detected

DBA_SS_BANK_NOT_DEFINED
An attempt has been made to store a bank which has not been defined in the calling routine's BOS array

DBA_SS_BANK_ALREADY_THERE
An attempt has been made to define or store a bank which is already on the database

DBA_SS_BANK_NOT_THERE
An attempt has been made to list or delete a bank on the dataset which is not there

DBA_SS_TOO_MANY_BANKS
There are more banks saved on the database than the maximum defined by the list and delete routines

DBA_SS_INVALID_NUMBER
An attempt has been made to define, add, delete, or list a bank with bank number less than zero

DBA_SS_LENGTH_DIFFERENT

Either:	a bank has been defined with a length different to that given in the calling sequence
Or:	a bank has been replaced on the database with length different to that for the last bank saved with the same name

DBA_SS_LOCKED
The database is locked by another user

A.3 Severe Error Detected

DBA_SS_ARRAY_TOO_SMALL

A bank cannot be defined or read because the calling routine's BOS array is too small

DBA_SS_NOT_INIT

An attempt has been made to call the database access routines without first calling DBA_INIT_PACKAGE

6. Appendix B An Example

```
      PROGRAM EXAMPLE
      IMPLICIT NONE
      INCLUDE 'A_DBA$DIR:DATABASE.INC'
C
C DEFINE BOS ARRAY
C
      COMMON/BCS/IW (20000)

      INTEGER STATUS, NUMBER, OUTPUT_LENGTH, INDEX
      INTEGER NAMES(500),NUMBERS(500),WORDS(500)
      INTEGER N_BANKS
      INTEGER BANK_NAMES (6)
      INTEGER BANK_NUMBERS (6)
      INTEGER TOTAL
      INTEGER INPUT_LENGTH
C
C INITIALISE DATABASE
C
      STATUS = DBA_INIT (92,0)
C
C DEFINE BANK
C
      INPUT_LENGTH = 40
      STATUS = DBA_DEFINE_BANK
     + (92, 'FRED',INPUT_LENGTH,NUMBER,OUTPUT_LENGTH,INDEX)
C
C FILL WITH DATA
C
      IW (INDEX + 1) = 1
      IW (INDEX + 2) = 2
      IW (INDEX + 3) = 3
```

```
C
C SAVE ON DATABASE
C
            STATUS = DBA_REPLACE_BANK (92, 'FRED', NUMBER)
C
C DEFINE BANK WITH NBANK, WITH NUMBER = 99 AND LENGTH = 400
C
            STATUS = DBA_DEFINE_NEW_BANK (92, 'BILL', 99, 400, INDEX)
C
C ADD TO DATABASE SPECIFYING A FORMAT
C
            STATUS = DBA_INCLUDE_BANK (92, 'BILL', 99, '2I,A,(F)')
C
C LIST BANKS
C
            STATUS = DBA_LIST_ALL_BANKS
            +           (92, 100, NAMES, NUMBERS, WORDS, TOTAL)

            DO I = 1,TOTAL
            TYPE  *, 'NAME ',NAMES (I),'NUMBER',NUMBERS(I)
            END DO
C
C SEND MESSAGE REQUESTING 'FRED' AND 'BILL' TO BE WRITTEN TO TAPE
C
            N_BANKS = 2
            BANK_NAMES (1) = 'FRED'
            BANK_NUMBERS (1) = NUMBER
            BANK_NAMES (2) = 'BILL'
            BANK_NUMBERS (2) = 99

            STATUS = DBA_WRITE_BANKS_TO_TAPE
           +           (N_BANKS,BANK_NAMES,BANKS_NUMBERS)
C
C REMOVE BANK 'BILL'  FROM THE DATABASE
C
            STATUS = DBA_REMOVE_SINGLE_BANK (92, 'BILL', 99)
```

```
C
C GARBAGE COLLECT THE DATABASE
C
      STATUS = DBA_GARBAGE_COLLECT  (92)

      END
```