DELPHI Collaboration **DELPHI 86 − 28 PROG − 46**

CARGO − DBMS **8 July 1986**

CARGO
A HIERARCHICAL DATA − BASE MANAGEMENT SYSTEM

A TECHNICAL MANUAL

CARGO is a DataBase Management System developed for the storage and access of data of the Hierarchical Structured type. The design has been worked out by F. Carena, G. Gopal and V. Perevozchikov, in response to requests from many people in DELPHI for such a generalised System. The implementation of this DBMS has been done by F. Carena (CERN) and G. Gopal (RAL). Any questions/comments on this System are very welcome. This document has been written by

G. P. Gopal (RAL)

CONTENTS

TABLES

1. INTRODUCTION

Until now Particle Physics Experiments have used a large number of ways of storing any data necessary for the acquisition (on-line) and processing (off-line) of events. These have varied from the simple use of DATA statements buried in the program code or the use of more or less complicated assignment statements during program initialization, to sophisticated uses of external files containing the necessary information accessed by program at execution time. In fact, even within a single experiment a common method has not been adopted for use in the numerous processing tasks. This has functioned reasonably well in the past as experiments have been generally small, run by small groups of people, with essentially all the software being centrally written and maintained. However, in the LEP era none of these features apply to the proposed experiments. The experiments, e.g. DELPHI, have the following features which necessitate the use of a central common method of data storage and access: –

- size (both the hardware & the collaboration)
- complexity
- long-life
- build-up over many months
- probability of frequent change
- modularity

and both the on-line and off-line software, needing to access the relevant data for control and analysis will have the following relevant features:

- designed & written by many people at many institutes, i.e. distributed programming
- to run on several types of main-frames
- built-up from many independent modules
- ability to run in interactive mode with Graphics display facilities as an integral part
- must be user-friendly to allow ease of execution by each & every member (almost) of a large Collaboration.

These features force the use of a central Data Base Management System, which would allow storage of different types of data on many different files without any specific restrictions. Commercial Data Base Management Systems are usually designed to have this capability, but have the following drawbacks:

- require large resources (CPU and Managerial)
- cost real money to purchase and maintain.
- learning process required before data structures can be set-up and data entered.
- write and read access times are suited to commercial applications and not Physics Experiments.

We feel that Commercial DataBase Management Systems have serious drawbacks and have learnt from experience that ad-hoc Systems designed to meet specific needs are not general enough to meet all the DataBase requirements of a Physics Experiment. We have therefore worked out a scheme for a general DataBase Management System (of the Hierarchical Structured type) for use in DELPHI based on the "Key Word Access Package" – KAPACK [3] with its excellent features – e.g. allowing variable length direct access records and providing access via a well-defined and flexible naming convention – as the basic tool. We choose the hierarchical structured type as this, we feel, is best suited to satisfy the constraints imposed by modularity (both in hardware and software) and to meet the requirement of fast read and write access.

The terminology used in describing the proposed System is given in the Glossary in Appendix 1. In Chapter 2 the file structure and how it can be built-up is described. The data structure in any particular data file is set-out in user-defined "Trees". Each "Tree" structure is made of Records of one or more Types. A "Record-Type" in a "Tree" is completely defined only when the structure of the data contained in records of this type is defined. This is done by defining the various "Fields" of data within the record. How the complete structure of user Data Records in a given Data File is defined is explained in Chapter 3. The methods used for the storage and access of user Data – either complete Records or Fields within a Record – for any Data File are described in Chapter 4. The storage of the Data structure definitions is done in "System" Records. Chapter 5 describes the layout and format of these.

A principal requirement of a DBMS for use in a Physics Experiment in which many geographically separated institutes collaborate is the ability to import/export part of the DataBase, or the complete DataBase from machine to machine. An extensive set of routines have been written to cater for this requirement. These are described in Chapter 6.

At each level of the description of this Hierarchical Data Base Management System the basic routines used to provide the necessary functionality are described. These routines form the Basic Package of the DBMS (DBBASP). Not all of the routines in this package are needed by all the users. The System Manager and each Local File Manager are the only people who will make use of all of these routines and even then through the interactive Control, Enquiry and Modification Package (DBCEMP) – necessary to make the DBMS work in a user-friendly manner . This package is described in Chapter 7. Finally, an example of an application of this DBMS – the storage and access of the Detector Alignment Constants is given in Reference [2]

2. DBMS FILE STRUCTURE

A user who has identified a set of data that are going to be frequently accessed by one or more programs can open a Data File in which this data can be stored. The complete structure of the data has to be entered and will be stored in "System Records" in this file. So the structure of the data is not hard-wired in any DBMS program code. A "System File" keeps track of all the Files in the DataBase.

2.1 System File

The function of this File is to contain information about all the "Data Files" in the DataBase and store the DataBase Password and Title. An interactive user or program is allowed full access to this File and all the Data Files if the overall Password is declared. In a program it is done by a call to routine DBOPEF (see section 2.1.3 below) at intialisation time. The information contained in this File is the following :

1. the Data Base Title,
2. the DataBase Password,
3. the number of Data Files and for each
    a. the Logical Unit Number,
    b. the Local Password,
    c. the File Title and
    d. the declared maximum size.

The Passwords and the File Titles are Character Strings of, in principle, any length. However, the requirement (FORTRAN77) that the length of a Character Variable be predefined has meant that there is an upper limit of 60 Characters.

The set of functions relating to this File and the routines to perform them are :

1. DBMAKE: to create the System File by entering a logical unit number associated with it, a global Title for the DataBase and a DataBase Password.
2. DBINIT: to declare the Logical Unit No. of the System File if the DataBase already exists. The Basic DBMS Package can then prepare in its local area the information necessary for any subsequent operation on any File in the DataBase.
3. DBOPEF: to allow write-access to all Files in the DataBase.The correct System Password results in full access to all the Files in the DataBase. This routine is also used for gaining full access to a single Data File by giving the correct Local Password.
4. Routines to alter
    a. the Global Title of the DataBase or the local Title of any given Data File – routine DBRFLT.

b. the System (Global) Password or the Local Password of any given Data File – routine DBRFLP.

c. the current debug printing option on (Default='OFF') for any File – routine DBRFDB.

d. the DBMS operations printed output unit (Default=6) for any File – routine DBRFPR.

5. Routines to fetch from the System File any of the following information about any of the Data Files in the DataBase.

   a. DBFILU: Logical Unit numbers in use for the Data Files in the DataBase.

   b. DBFILT: the File Title for any File (Data or System).

   c. DBFILS: the declared maximum size of a Data File and the fraction of it currently used.

   d. DBGFDB: the current debug option for a given File.

   e. DBGFPR: the logical unit number of the file to which error/debug messages are currently being routed.

   Note : A different logical unit may be used for debug printing for each Data File in the DataBase.

The routines for the functions 1., 3. and 4. above require full access to the System and/or the appropriate Data File.

2.1.1 DBMAKE – System File Operations

```
CALL DBMAKE (LUNSYS,DBTIT,SYSPAS,IERROR)
```

This routine is called at the start of the setting-up of the DataBase. It opens a Direct Access System File which will be used to control the Data Files.

Input:

LUNSYS	Logical Unit No. of the System File $(0 < LUNSYS < 100)$
DBTIT	The DataBase Title Character String of any length
SYSPAS	The Global DataBase Password Character String of any length

Output:

IERROR	Error Flag = 1, if a File is already associated with unit LUNSYS. = 2, Buffer overflow – Title/Password too long.

2.1.2 DBINIT

```
CALL DBINIT (LUNSYS,IERROR)
```

The routine loads into local memory, reserved for the purpose by the DBMS Basic Package, all the information stored on the System File for each of the Files in the DataBase. The routine checks that declared System File has consistent information about each of the Data Files known to it.

Input:

LUNSYS	Logical Unit No. of the System File.

Output:

IERROR	Error Flag = 1, if LUNSYS is incorrect, or System File does not exist, = 2, if Logical error on a Data File, = 3, if there is a mis-match between the System File and a Data File.

2.1.3 DBOPEF

```
CALL DBOPEF (LUN,PASSW,IERROR)
```

This routine is used to get write-access to any File (Data or System) in the DataBase.

Input:

LUN	Logical Unit No. of the File.
PASSW	Password associated with the File.

Output:

IERROR	Error Flag = 1, if Logical Unit number is not known to the System File, = 2, if incorrect Password given, only Read Access given.

2.1.4 DBRFLT

```
CALL DBRFLT (LUN,FILTIT,IERROR)
```

This routine to be used to change the File (System or Data) Title.

Input:

LUN	Logical Unit No. of the File.
FILTIT	the new File Title (any length).

Output:

IERROR	Error Flag = 1, Illegal Unit Number. = 2, Buffer Overflow – Title too long. = 3, write-access not available.

2.1.5 DBRFLP

```
CALL DBRFLP (LUN,PASSW,IERROR)
```

This routine to be used to change the File (System or Data) Password.

Input:

LUN	Logical Unit No. of the File.
PASSW	the new File Password (any length),

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Buffer Overflow – Password too long. = 3, write-access not available.

2.1.6 DBRFDB

```
CALL DBRFDB (LUN,ION,IERROR)
```

This routine to be used to change the File (System or Data) Debug Printing option.

Input:

LUN	Logical Unit No. of the File.
ION	the new Print-option (=0, OFF & =1, ON) (Default = OFF)

Output:

IERROR	Error Flag = 1, Illegal unit number.

2.1.7 DBRFPR

```
CALL DBRFPR (LUN,LUNPR,IERROR)
```

This routine to be used to change the unit for printed output from KAPACK (default = 6) for any operations on the File (System or Data).

Input:

	LUN	Logical Unit No. of the File.
	LUNPR	the unit no. for printed output.

Output:

	IERROR	Error Flag = 1, Illegal unit no. for DB File. = 2, Illegal unit no.for Printed output

2.1.8 DBFILU

```
CALL DBFILU (NMAX,NU,LUNITS,IERROR)
```

The routine returns the logical unit number associated with each Data File in the DataBase.

Input:

	NMAX	Maximum number of Files expected to be in the DataBase.

Output:

	NU	The Number of Data Files in the DataBase
	LUNITS	Array containing the logical unit numbers associated with each File. Note: If NU is greater than NMAX then only the logical unit numbers of the first NMAX Files are returned.
	IERROR	Error Flag = 1, NU > NMAX.

2.1.9 DBFILT

```
CALL DBFILT (LUN,MAXLEN,NCHAR,FILTIT,IERROR)
```

The routine returns the Local File Title associated with a given Data File in the DataBase. The DataBase Title is returned if LUN is the logical unit number of the System File.

Input:

LUN	Logical Unit No. of the File.
MAXLEN	Expected Maximum length (no. of Characters) of the Title

Output:

NCHAR	The length (no. of Characters) of the Data File Title.
LOCTIT	Character string containing the Title associated with the File. Note: If NCHAR is greater than MAXLEN then only the first MAXLEN characters in the File Title are returned.
IERROR	Error Flag = 1, Illegal unit no. = 2, NCHAR > MAXLEN.

2.1.10 DBFILS

```
CALL DBFILS (LUN,ISIZE,IERROR)
```

The routine returns the declared maximum size of a given File (System or Data) and the percentage currently used.

Input:

LUN	Logical Unit No. of the File.

Output:

ISIZE	Array with 2 elements : ISIZE (1) = Maximum no. of Records, ISIZE (2) = Average length of each. ISIZE (3) = %age fraction currently used.
IERROR	Error Flag = 1, Illegal unit no.

2.1.11 DBGFDB

```
CALL DBGFDB (LUN,ION,IERROR)
```

This routine is used to find out the current Debug Printing option for a File (System or Data) .

Input:

	LUN	Logical Unit No. of the File.

Output:

	ION	the current Print-option (=0, OFF & =1, ON)
	IERROR	Error Flag = 1, Illegal unit number.

2.1.12 DBGFPR

```
CALL DBGFPR (LUN,LUNPR,IERROR)
```

This routine to be used to fetch the unit no. used for printed output from KAPACK (default = 6) for any operations on a File (System or Data).

Input:

	LUN	Logical Unit No. of the File.

Output:

	LUNPR	the unit no. for printed output.
	IERROR	Error Flag = 1, Illegal unit no. for DB File.

2.2 Data Files

Needless to say, there must be one or more Data Files in a DataBase apart from the System File. These are Direct Access Files used to store user data. Since the aim is to achieve as much flexibility as possible a complete description of the user data (contained in a given Data File) is also stored in this File. For security purposes the originator/manager of each Data File has to define a unique Local Password which is stored in the File itself and in the System File as well to allow the overall System Manager complete access to the whole DataBase. Opening the File with routine DBOPEF (defined above) gives the user full access to it.

The set of functions relating to an individual Data File and the routines to perform them are :

1. DBADDF: to add a new Data File to the DataBase. The Logical unit number, its maximum size, the File Title and the Local Password have to be given. This information has first got to be entered into the System File before the File can be opened. Therefore a knowlegde of the DataBase Password is necessary before the DBMS Basic Package will perform the function. A call to routine DBOPEF to open the System File with write-access is necessary before calling routine DBADDF.

2. DBDELF: to remove a Data File from the DataBase. The logical unit number associated with the File is removed from the list of Logical Unit Numbers stored in the System File. The File is not physically deleted, it only becomes inaccessible to the DBMS. As for addition of a new Data File, write access to the System File must have been obtained by a call to DBOPEF giving the correct DataBase Password before calling routine DBDELF.

3. DBEXPF: to increase the declared maximum size of an existing Data File. This function is performed such that at the end of it the DataBase should look identical as before except for the size of the requested File having been increased. It is necessary, therefore, for the DBMS to copy the original to a scratch unit, delete the file from the DataBase, add a new expanded File with all other characteristics unchanged and then copy the original File from the scratch unit to the appropriate File.

2.2.1 DBADDF – Data File Operations

```
CALL DBADDF (LUN,ISIZE,FILTIT,LOCPAS,IERROR)
```

The routine enters information about a new Data File to the System File and opens the new Data File. The same information is also entered in the File itself.

Input:

LUN	Logical Unit No. of the Data File.
ISIZE	Array with two elements ISIZE (1) = Expected number of Records and ISIZE (2) = Average Record Length
FILTIT	The File Title Character String of any length N.B.:– This is only meaningful in the DB Context. It must not be confused with machine-dependent File name.
LOCPAS	The Local Password, a Character String of any length.

Output:

IERROR	Error Flag = 1, LUN is already defined in DataBase. = 2, No. of Data Files limit hit. = 3, Invalid Size given. = 4, Another KAPACK File with same unit no. = 5, Buffer Overflow – FILTIT/LOCPAS too long. = 6, write-access to System File not available.

2.2.2 DBDELF

```
CALL DBDELF (LUN,IERROR)
```

This routine removes all reference, in the System File, to the Data File associated with the given Logical Unit Number. Write-access to the System File must be obtained prior to calling this routine.

Input:

LUN	Logical Unit No. of the Data File.

Output:

IERROR	Error Flag = 1, Unit no. LUN unknown to DataBase. = 2, System File cannot be deleted. = 3, write-access to System File not available.

2.2.3 DBEXPF

```
CALL DBEXPF (LUN,LUNSCR,ISIZE,IERROR)
```

Routine to increase the number of blocks allocated to a Data File. A Scratch File has to be assigned by the user to perform this function.

Input:

LUN	Logical Unit No. of the Data File.
LUNSCR	Logical Unit No. of the Scratch File.
ISIZE	Array with two elements giving the new size ISIZE (1) = Expected number of Records and ISIZE (2) = Average Record Length

Output:

IERROR	Error Flag = 1, Unit no. LUN not know to DataBase. = 2, Illegal unit no. for Scratch File. = 3, Invalid size. = 4, write-access to Data File not obtained.

3. STRUCTURE OF DATA RECORDS IN DATA FILES

Before user data can be entered in records in the defined user Data File, the complete structure of the records must be known to the DBMS. It is stored in System Records – see Chapter 5 – in the Data File itself. The structure must be built-up in a hierarchical manner in the form of defined "Trees" of records.

3.1 Data Structure definition – "Trees"

As defined in the Glossary a "Tree" is made up of a set (one or more) of Records linked by a "Root-Branch" or a "Parent Branch-Branch" Relationship. Each record must have at least one "Root" or a "Parent Branch" and can have any number of "sub-Branches". For each Tree-Type there must be at least one Record-Type which is defined as a Parent/Root Record-Type.

In a given Data File there can be any number of different Tree-Types, each identified by:

1. A Title – a character string of any length for a readable description of the Tree.
2. A Code – a string of 4 Characters identifying a Tree-Type for the DBMS structure defining routines.

A workable system of defining, changing, removing different types of Trees and altering their attributes is required in any Hierarchical DBMS. For this purposes routines have been defined to perform the following functions:

1. DBADTT: to enter a new Tree-Type giving a short explanation and its identification code. These are entered in a "Tree Defining System Record" – part of the system records that will contain the Data Structure definitions for all user Data Records – in the given Data File.
2. DBDLTT: to remove from the list of Tree-Types in a given Data File an incorrectly defined, or no longer necessary, Tree-Type. This function is particularly necessary at the setting-up stage when a Data File is being prepared for subsequent storage of user Data by the Local Data File Manager (as opposed to the overall DataBase Manager). Any attempt to delete a defined Tree-Type when user Data already exists in the Data File in Records of this Tree-Type cannot be allowed as it would lead to a defined Data Structure unable to provide any information on existing user Data Records in the File. In this case it would be necessary to delete all Records belonging to this Tree-Type before it could be removed from the list of valid Tree-Types.
3. DBRTTT: to change the Title associated to an existing Tree-Type. This can be done despite the presence of user Data Records belonging to this Tree-Type as the Tree-Type Title is only a short explanation of the Tree-Type and has nothing to do with the data structure as such.
4. DBRTTC: to change the Code associated to an existing Tree-Type. The Code is used in the naming covention (see later) of user Data Records and therefore forms an integral part of the Data Structure definition. As such it cannot be altered if user Data Records belonging to the Tree-Type already exist before altering the names of all the associated Records.
5. DBTRTC: to provide a list of defined Tree-Types, i.e. the Tree-Type Codes in a given Data File.
6. DBTRTL: to get the Tree-Type Title associated with a particular Tree-Type.

It should be noted that full access to the Data File concerned is necessary – by a call to routine DBOPEF (see section 2.1.1) – for any of the functions 1. to 4. above.

3.1.1 DBADTT – Data Structure – Tree Level

```
CALL DBADTT (LUN,TRCODE,TRETIT,IERROR)
```

Routine to enter a new Tree-Type definition in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
TRETIT	the Tree Title – Character String of any length.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Illegal Tree-Type Code. = 3, Tree-Type already defined. = 4, Buffer Overflow in File header record. = 5, Buffer overflow in Tree-Type record. = 6, Read-access only to File.

3.1.2 DBDLTT

```
CALL DBDLTT (LUN,TRCODE,IERROR)
```

Routine to remove an existing Tree-Type from the list of Trees.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Tree-Type. = 3, Data Records already present. = 4, Read-access only to File.

3.1.3 DBRTTT

```
CALL DBRTTT (LUN,TRCODE,TRETIT,IERROR)
```

Routine to change the Title associated to an existing Tree-Type in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
TRETIT	the new Tree Title – Character String of any length.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Tree-Type Code unknown. = 3, Buffer Overflow – Title too long. = 4, Read-access only to File.

3.1.4 DBRTTC

```
CALL DBRTTC (LUN,TRCODO,TRCODN,IERROR)
```

Routine to change the Code associated to an existing Tree-Type in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODO	the Old Tree-Type Code – a 4-Character String.
TRCODN	the New Tree-Type Code – a 4-Character String.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Tree-Type Code unknown. = 3, New Tree-Type Code already used. = 4, New Tree-Type Code illegal. = 5, Data Records already present. = 6, Read-access only to File.

3.1.5 DBTRTC

```
CALL DBTRTC (LUN,NMAX,NTTYP,TRCODS,IERROR)
```

Routine to return a list of all Codes associated to existing Tree-Types in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
NMAX	Expected Maximum number of Tree-Types in the list.

Output:

NTTYP	Actual number of Tree-Types in Data File
TRCODS	CHARACTER*4 Array containing the list of Tree-Type Codes. Note :– When NTTYP > NMAX the Codes of the first NMAX Tree-Types are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, NTTYP > NMAX.

3.1.6 DBTRTL

```
CALL DBTRTL (LUN,TRCODE,NMAX,NCHAR,TRETIT,IERROR)
```

Routine to return the Title associated to an existing Tree-Type in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
NMAX	Expected Maximum number of Characters in the title.

Output:

NCHAR	Actual number of Characters in the Title.
TRETIT	Tree-Type Title – Character String containing the Tree Title. Note :– When NCHAR > NMAX the first NMAX Characters of the Title are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Tree-Type Code unknown. = 3, NCHAR > NMAX.

3.2 Data Structure definition – "Record-Types" in Trees

The definition, at the next level, of the user Data to be stored in a given Data File is that of all the Record-Types belonging to each type of Tree in the Data File. A Record-Type definition consists of

1. a Record-Type Code to differentiate types of Records in the same Tree,
2. a list of possible Parent Type Records,
3. a Record-Type Title giving a short explanation of the type of Data stored in the Record.

When a Record-Type is thus defined, it is added to the corresponding Tree-Type definition. The functions required at this level of structure definition are similar to those at the Tree level. The routines available are :

1. DBADRT: to add a new Record-Type to a given Tree-Type.
2. DBDLRT: to remove an existing (incorrectly defined or redundant) Record-Type from a given Tree-Type. This function can be performed only if no user Data records of this type exist in the Data File.
3. DBRRTT: to change a Record-Type Title. Since the Title is not part of the data structure definition a change can be effected at any time (even if user Data Records of this type are present).
4. DBRRTC: to rename the Record-Type, i.e. change the Record-Type Code. This can only be done when there are no user Data Records of this type present in the Data File. The code is used in the Record naming convention (see later).
5. DBRRTP: to change the structure associated to a Record-Type by altering the number and/or the list of Parent Record-Types. Again this function cannot be performed if user Data Records of this Type already exist.
6. DBRTYP: to get a list of all the declared Record-Types in a given Tree-Type.
7. DBRTTL: to get the Title associated with a particular Record-Type in a given Tree-Type.
8. DBRTLI: to get a list of possible Parent/Branch Record-Types associated to a given Record-Type in the Tree.

The functions 1. to 5. above define the Tree Structure and result in modifications to , or additions of, System Records in the given Data File. As such full access to the Data File is required to be able to perform them. The last three functions are of the query type and can be performed with read-access only.

3.2.1 DBADRT – Data Structure – Record Level

```
CALL DBADRT (LUN,TRCODE,RTCODE,RECTIT,NP,PARENT,IERROR)
```

Routine to enter a new Record-Type definition to a defined Tree-Type in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
RECTIT	the Record-Type Title – Character String of any length.
NP	the number of Parent Record-types to this Record-Type.
PARENT	Character*4 Array containing the list of possible Parent Record-Type Codes. Note: NP = 0 indicates the Record-Type to be a "Root" Record in the Tree.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Tree-Type not defined. = 3, Illegal Record-Type Code. = 4, Illegal Parent Record-Type. = 5, Record-Type already defined. = 6, Buffer overflow in TT Record. = 7, Buffer overflow preparing RT Record. = 8, Buffer overflow updating RT Record. = 9, Read-access only to File.

3.2.2 DBDLRT

```
CALL DBDLRT (LUN,TRCODE,RTCODE,IERROR)
```

Routine to remove an existing Record-Type definition in a given Tree-Type in a Data File – only when there are no user Data Records of this type present in the File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.

Output:

IERROR	Error Flag = 1, Illegal unit number.

	= 2, Unknown Record-Type.
	= 3, Data Records already present.
	= 4, Read-access only to File.

3.2.3 DBRRTT

```
CALL DBRRTT (LUN,TRCODE,RTCODE,RECTIT,IERROR)
```

Routine to change the Title associated to an existing Record-Type in a given Tree-Type in a Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
RECTIT	the new Record-Type Title – Character String of any length.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Record-Type.
	= 3, Buffer overflow.
	= 4, Read-access only to File.

3.2.4 DBRRTC

```
CALL DBRRTC (LUN,TRCODE,RTCODO,RTCODN,IERROR)
```

Routine to change the Code associated with an existing Record-Type in a given Tree-Type in a Data File – a change that can be done only if no user Data-Records of this type exist in the File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODO	the old Record-Type Code – a 4-Character String.
RTCODN	the new Record-Type Code – a 4-Character String.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Record-Type.
	= 3, New Code already defined.
	= 4, Illegal New Code.

= 5, Data Records already present.
= 6, Read-access only to File.

3.2.5 DBRRTP

```
CALL DBRRTP (LUN,TRCODE,RTCODE,NP,PARENT,IERROR)
```

Routine to change the number and Parent Record-Type names associated to an existing Record-Type in a given Tree-Type in a Data File – a change that can be done only if no user Records of this type exist in the File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
NP	the new number of Parent Records to this Record Type.
PARENT	Character*4 Array containing the new list of possible Parent Record-Type Codes. Note: NP = 0 will change the Record-Type to a "Root" Record in the Tree.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Record-Type.
	= 3, Illegal Parent Record-Type.
	= 4, Buffer Overflow.
	= 5, Data Records already present.
	= 6, Read-access only to File.

3.2.6 DBRTYP

```
CALL DBRTYP (LUN,TRCODE,NMAX,NRTY,RTCODS,IERROR)
```

Routine to return the number and Codes of Record-Types in a given Tree-Type in a Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
NMAX	expected number of Record-Types in the Tree.

Output:

NRTY	the actual number of Record-Types

RTCODS	Character*4 Array containing the list of Record-Type Codes. Note: If NRTY > NMAX the first NMAX Record-Type codes in the Tree are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Tree-Type. = 3, NRTY > NMAX.

3.2.7 DBRTTL

```
CALL DBRTTL (LUN,TRCODE,RTCODE,NMAX,NCHAR,RECTIT,IERROR)
```

Routine to get the Title associated to an existing Record-Type in a given Tree-Type in a Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
NMAX	expected no. of characters in the Record Title.

Output:

NCHAR	actual no. of characters in the Record Title.
RECTIT	the Record-Type Title – Character String of length = NMAX. Note : If NCHAR > NMAX the first NMAX characters in the Record Title are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Record-Type. = 3, NCHAR > NMAX.

3.2.8 DBRTLI

```
CALL DBRTLI (LUN,TRCODE,RTCODE,IUPDOW,NMAX,NC,CODES,IERROR)
```

Routine to return the number and Parent/Son Record-Type Codes associated to an existing Record-Type in a given Tree-Type in a Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.

	IUPDOW	flag = 1 for Parent Record-Types, = 2 for Son Record-Types.
	NMAX	expected no. of Parent/Son Record-Types.
Output:		
	NC	actual no. of requested Record-Type Codes.
	CODES	Character*4 Array containing the list of requested Record-Type Codes. Note : If NC > NMAX the first NMAX Record-Type codes in the list are returned.
	IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Record-Type. = 3, NC > NMAX.

3.3 Data Structure definition – "Fields" in Records.

Each Record containing user Data is made up of "Fields", the first of these is the Header Field which is fixed by the DBMS. The user provides the start date and time of a period of validity of the user data contained in the record. The user data within the body of the Record can be structured in one or more defined Fields. Access to user data is allowed at two levels in CARGO – the complete Record or a given named/numbered Field within the Record.

Each user defined Field in a record is given a 4-Character Name ('HEAD' is reserved for the header Field) and is assigned a unique number. The first word in the Field is reserved for its length (the no. of words following) and, therefore, has to be of type Integer (I). The following three types of Fields are considered sufficient:

1. Fixed Length Field: Length of the Field (in number of 4-byte words) is fixed for all Records of a given type. It is declared at the time of setting up the Data Structure of user Records in the File. Also the Description – a line of text – and Data-Type (Alphanumeric (A), Integer (I), Real (R) or Bit-Mask (B)) of each word in such a Field have to be declared at the time of setting-up.

2. Variable Length Field: As the name suggests the length can vary from record to record for such Fields, being set when the user Data is entered. All the words in such Fields, following the first, will have the same (assigned during setting-up) Description and Data-Type

3. Coded Field: This is essentially a Fixed Length Field, but its length depends on a parameter (the Code-Name) stored in the Field itself. A finite number of possible values of this parameter have to be given at the time of declaration of a Field of this kind. Also the length associated with each value has to be entered. The second word in such a Field is of type Alphanumeric (A) and equals one of the declared possible Code-Names. This is followed by the Data words with fixed Descriptions and Data-Types declared to the DBMS for each Code-Name.

A declaration of a Field, therefore, requires the above quantities to be given to the DBMS for storage in the System Records in the Data File. It should be noted that the DBMS treats Fixed and Variable Length Fields in the same manner as a Coded Field with the restriction that the list of Code-Names is only one long. The single Code-Name for either of this type of Field is set equal to the Field-Name.

The routines available at this level are:

1. DBADFD: to define a Field.

2. DBDLFD: to remove a Field from the structural definiton of a Record. This can only be done while there are no Data Records of the type in the Data File.

3. DBRFDN: to change a Field Name. The Field Name is not used in any structural or naming capacity. This change can therefore be done even when Data Records are present.

4. DBRFDT: to change the Field Title. The Field Title, as the Name, is use for presentation purposes and thus can be altered at any time.

5. DBADCF: to enter a new possible Code-Name, the no. of words associated with it and the Data Type of each word for a defined Field. This can be done even when there are user Data Records of the given Type present since it does not alter the structural definition.

6. DBDCCF: to delete a particular declared Code-Name in a Field. If the particular Code-Name is actually used in any user Data-Records this function cannot be performed. To check the use of a particular Name in all user Data-Records is extremely costly. Therefore, routine DBDCCF will return an error condition if user Data-Records of the given Record-Type are present.

7. DBRCNA: to redefine a declared Code-Name in a Coded Field. Again the same restrictions apply as for the preceding function.

8. DBRCCF: to alter the Field length and/or the Data Formats of any, or all, of the words associated to a particular Code-Name in a Field. The restrictions mentioned for routine DBDCCF also apply to the use of this routine.

9. DBRCDT: to alter the Title associated to a particular Code-Name in a Field. This routine can be called even when user Data Records of this type are present since the Code-Title is not used in the structural definition or in a naming capacity.

The above defined functions are all necessary to establish correctly the complete structure of the Fields making up the Records of any type in any Tree in a Data Field. As such full-access to the Data File is required for these to take effect. Unless otherwise stated each of the above functions can only be performed if there are no user Data-Records of the given Type present in the Data File.

Besides these there is another set of functions that are necessary to be defined so that interactive input/access of user Data can be done in a manner meaningful to a casual user. This is the storage and retrieval of textual explanation of what each word in a particular Field in the Record means. Also, for certain applications it may be necessary to store for each word either minimum and maximum values or a set of discrete allowed values. A complete word description, therefore, is made up of all these quantities. The related functions and routines are :

1. DBAWDE: to enter a word description for a given word in a Field.

2. DBDWDE: to delete the word description of a given word in a Field.

3. DBRWDE: to alter the word description of a given word in a Field.

Again as these word descriptions are stored in System Records in the Data File full access to the File is essential.

The above sets of functions are all that are needed to complete the full definition and description of the structure of user Data Records in a Data File. However, to aid the local File Manager and help a casual user see the defined Data Structure a set of access/query functions (and routines) have been defined :

1. DBFDNA: to get a list of names of all Fields defined for a given Record-Type.
2. DBFDTY: to determine the Type of a named Field.
3. DBFDNU: to get the number corresponding to a Field Name.
4. DBFDTL: to get the Title assigned to a named Field.
5. DBCODS: to get a list of the declared possible Code-Names in a Field.
6. DBCFDL: to determine the length and data formats associated to a particular Code-Name in a given Field.
7. DBCDTL: to get the Title associated with a given Code Name in a Coded Field.
8. DBGWTL: to get the textual description associated to a given word in a Field.
9. DBGWRD: to get the range or the list of allowed discrete values for a given word in a Field.

3.3.1 DBADFD – Data Structure – Field Level

```
CALL DBADFD (LUN,TRCODE,RTCODE,FLDNAM,FLDTYP,FLDTIT,IERROR)
```

Routine to enter a new Field definition to a defined Record-Type in a given Tree-Type in a Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
FLDTYP	CHARACTER*1 (= F or V or C – for Fixed Length, Variable Length & Coded Fields resp.)
FLDTIT	Field Title – Character String of any length

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Record-Type not defined. = 3, Illegal Field Name. = 4, Field Name already defined. = 5, Illegal Field Type. = 6, Data Records of this type present. = 7, Buffer overflow replacing RT Record. = 8, Buffer overflow preparing Field Record. = 9, Read-access only to File.

3.3.2 DBDLFD

```
CALL DBDLFD (LUN,TRCODE,RTCODE,FLDNAM,IERROR)
```

Routine to delete a defined Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Field.
	= 3, Data Records of this type present.
	= 4, Read-access only to File.

3.3.3 DBRFDN

```
CALL DBRFDN (LUN,TRCODE,RTCODE,OFDNAM,NFDNAM,IERROR)
```

Routine to alter the name of a defined Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
OFDNAM	4-Character old Field Name.
NFDNAM	4-Character new Field Name.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Field.
	= 3, New Field Name already defined.
	= 4, New Field Name is illegal.
	= 5, Read-access only to File.

3.3.4 DBRFDT

```
CALL DBRFDT (LUN,TRCODE,RTCODE,FLDNAM,FLDTIT,IERROR)
```

Routine to alter the Title associated to a defined Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
FLDTIT	new Field Title – Character String (any length)

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Field. = 3, Buffer overflow. = 4, Read-access only to File.

3.3.5 DBADCF

```
CALL DBADCF (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,CODTIT,NW,
                  DATYP,IERROR)
```

Routine to add a new possible Code-Name to a defined Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name. Note : = FLDNAM for an F- or V-type Field.
CODTIT	Code Title – Character String (any length).
NW	no. of words for this Code-Name. Note : NW = 1 for a V-type Field.
DATYP	Character*NW string – the Data Types of all the words.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Field.
	= 3, Illegal Code Name.
	= 4, Length inconsistent with Field-Type.
	= 5, Illegal Data Format.
	= 6, Code Name already in use.
	= 7, Buffer overflow replacing Field Record.
	= 8, Buffer overflow preparing Code Record.
	= 9, Read-access only to File.

3.3.6 DBDCCF

```
CALL DBDCCF (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,IERROR)
```

Routine to delete an existing Code-Name in a defined Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Code Name.
	= 3, Data Records already present.
	= 4, Read-access only to File.

3.3.7 DBRCNA

```
CALL DBRCNA (LUN,TRCODE,RTCODE,FLDNAM,OCDNAM,NCDNAM,IERROR)
```

Routine to replace an existing Code-Name for a defined Coded Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
OCDNAM	4-Character old Code Name.
NCDNAM	4-Character new Code Name.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Code Name.
	= 3, New Code Name already in use.
	= 4, New Code Name – Illegal.
	= 5, Data Records already present.
	= 6, Read-access only to File.

3.3.8 DBRCCF

```
CALL DBRCCF (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NW,DATYP,
                    IERROR)
```

Routine to change the length and data formats associated with a declared Code-Name in a defined Field. Again, for an F- or V-type of Field the only Code-Name is the same as the Field-Name.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name.
NW	no. of words (new or old if the length is not being changed). (= 1, for a V-type Field.)
DATYP	Character*NW – the Data-Types for all words.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Code Name.
	= 3, Length inconsistent with Field-Type.
	= 4, Illegal Data Format.
	= 5, Data Records already present.
	= 6, Buffer Overflow.
	= 7, Read-access only to File.

3.3.9 DBRCDT

```
CALL DBRCDT (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,CODTIT,IERROR)
```

Routine to change the title associated with a declared Code-Name in a defined Field. For an F- or V-type of Field the Code-Title is not meaningful.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name.
CODTIT	Code Title – Character String (any length).

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Code Name. = 3, Buffer Overflow. = 4, Read-access only to File.

3.3.10 DBAWDE

```
CALL DBAWDE (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NW,RANGE,NDV,
                    ALIST,WDESC,IERROR)
```

Routine to add the description for a given word in a declared Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name.
NW	the word no. Note : for a V-type Field there is only a single word description applicable to all words (hence = 1)
RANGE	Array of 2 elements containing the minimum and maximum values. If the difference is less than or equal to 0.0, no limits are assumed.
NDV	the no. of allowed discrete values.
ALIST	Array containing the list.
WDESC	Textual description – Character String (any length)

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Code Name. = 3, Word Number Illegal. = 4, Range incompatible with existing Data Records. = 5, List of Discrete Values incompatible with existing Data Records. = 6, Buffer Overflow. = 7, Read-access only to File. = 8, Word description already exists.

3.3.11 DBDWDE

```
CALL DBDWDE (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NW,IERROR)
```

Routine to delete the description for a given word in a declared Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name.
NW	the word no. (= 1 for V-type Field).

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Read-access only to File. = 3, Word description does not exist.

3.3.12 DBRWDE

```
CALL DBRWDE (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NW,RANGE,NDV,
                        ALIST,WDESC,IERROR)
```

Routine to change the description for a given word in a declared Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code Name.
NW	the word no. (= 1 for a V-type Field).
RANGE	Array of 2 elements containing the minimum and maximum values. If the difference is less than or equal to 0.0, no limits are assumed.
NDV	the no. of allowed discrete values.
ALIST	Array containing the list.
WDESC	Character String containing the new Description.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Code Name. = 3, Word Number Illegal. = 4, Range incompatible with existing Data Records. = 5, List of Discrete Values incompatible with existing Data Records. = 6, Buffer Overflow. = 7, Read-access only to File. = 8, Word description does not exist.

3.3.13 DBFDNA

```
CALL DBFDNA (LUN,TRCODE,RTCODE,NMAX,NF,FDNAMS,IERROR)
```

Routine to return a list of declared Field Names in a given Record.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
NMAX	Expected no. of Fields.

Output:

NF	actual number of Fields in Record-Type.
FDNAMS	Character*4 array of Field Names Note : If NF > NMAX then the names of the 1st NMAX Fields are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Record-Type. = 3, NF > NMAX.

3.3.14 DBFDTY

```
CALL DBFDTY (LUN,TRCODE,RTCODE,FLDNAM,FLDTYP,IERROR)
```

Routine to determine the Type of a given Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.

Output:

FLDTYP	Character*1 – Field-Type (F, V or C).
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Field.

3.3.15 DBFDNU

```
CALL DBFDNU (LUN,TRCODE,RTCODE,FLDNAM,NUF,IERROR)
```

Routine to get the sequential number of a field of a given name.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.

Output:

NUF	Sequential Field Number.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Field.

3.3.16 DBFDTL

```
CALL DBFDTL (LUN,TRCODE,RTCODE,FLDNAM,NMAX,NCHAR,FLDTIT,
                    IERROR)
```

Routine to get the Title associated to a given Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
NMAX	Expected length (no. of Characters) of the Title.

Output:

NCHAR	actual no. of characters in the Title.
FLDTIT	Character*NMAX String containing the Title. Note : If NCHAR > NMAX the first NMAX Characters in the Title are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Field. = 3, NCHAR > NMAX.

3.3.17 DBCODS

```
CALL DBCODS (LUN,TRCODE,RTCODE,FLDNAM,NMAX,NC,CODNAM,
                    IERROR)
```

Routine to get a list of possible Code-Names for a particular Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
NMAX	expected no. of Possible Code-Names.

Output:

NC	actual no. of Code-Names.
CODNAM	Character*4 Array containing the Code-Names. Note : IF NC > NMAX then the 1st NMAX Code-Names are returned.

IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Field Name. = 3, NC > NMAX.

3.3.18 DBCFDL

```
CALL DBCFDL (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NMAX,NW,DATYP,
                     IERROR)
```

Routine to return the Length and the Data Formats of all the words associated with a particular Code-Name in a given Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code-Name.
NMAX	expected Length of the Field for this Code.

Output:

NW	actual Length of the Field for this Code-Name.
DATYP	Character*NMAX Array containing the Data Types. Note : IF NW > NMAX then the Data Types for the 1st NMAX words are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Code Name. = 3, NW > NMAX.

3.3.19 DBCDTL

```
CALL DBCDTL (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NMAX,NCHAR,
                     CODTIT,IERROR)
```

Routine to return the Title associated with a particular Code-Name in a given Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code-Name.

NMAX	expected Length of the Code Title.

Output:

NCHAR	actual Length of the Code-Title.
CODTIT	the Code Title (CHARACTER*NMAX). Note : IF NCHAR > NMAX then the 1st NCHAR characters are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Unknown Code Name. = 3, NCHAR > NMAX.

3.3.20 DBGWTL

```
CALL DBGWTL (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NW,NMAX,NC,
                     WDESC,IERROR)
```

Routine to return the textual description for a particular word in a Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code-Name.
NW	the word no. in the Field. Note : (= 1) for a V-type Field.
NMAX	expected Length of the Description.

Output:

NC	actual no. of Characters in the Line Description.
WDESC	Character*NMAX Array containing the Line Description. Note : IF NC > NMAX then the 1st NMAX Characters in the Description are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Word description does not exist. = 3, NC > NMAX.

3.3.21 DBGWRD

```
CALL DBGWRD (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NW,NMAX,RANGE,
                  NDV,ALIST,IERROR)
```

Routine to return the range or the set of allowed discrete values associated to a particular word in a Field.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	the Record-Type Code – a 4-Character String.
FLDNAM	4-Character Field Name.
CODNAM	4-Character Code-Name.
NW	the word no. in the Field. Note : (= 1) for a V-type Field.
NMAX	expected no. of allowed discrete values.

Output:

RANGE	Array containing minimum and maximum values.
NDV	actual no. of allowed discrete values.
ALIST	Array containing the defined allowed values. Note : IF NDV > NMAX then the 1st NMAX values are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Word description does not exist. = 3, NDV > NMAX.

3.4 Interrogating the Structure

Once a Data File has been entered with a complete definition of the structure of each type of Data-Record envisaged, access (read/write) to Data-Records can occur. The user will, from time-to-time, find it necessary to remind himself of the defined structure to make meaningful use of the DataBase. For this purpose a special routine – DBPRDS – is provided. It searches for the complete Data-Structure defined for a given Data File from any give level (File/Tree/Record/Field etc.) downwards and lists it on the printing unit reserved for the given Data File (see DBRFPR – 2.1.7 above). The routine is described below :

3.4.1 DBPRDS – Data-Structure Interrogation

```
CALL DBPRDS (LUN,TRCODE,RTCODE,FLDNAM,CODNAM,NWN,LEVEL1,
                  LEVEL2,IERROR)
```

Routine to list the defined Data-Structure for a given File/Tree-Type/Record-Type/Field.

Input:

LUN	Logical Unit No. of the Data File. = 0, all Data Files.
TRCODE	Tree-Type Code (CHARACTER*4) = '****', all Tree-Types.
RTCODE	Record-Type Code (CHARACTER*4) = '****', all Record-Types.
FLDNAM	Field Name (CHARACTER*4) = '****', all Fields.
CODDNAM	Code Name (CHARACTER*4) = '****', all Codes.
NWN	Word Number in Field. = 0, all words.
LEVEL1	Start Level for detailed printout.
LEVEL2	End Level. Each Structure Record is printed in detail depending on the Range : LEVEL1 to LEVEL2. LEVEL = 1 for File, upto LEVEL = 6 for Words.

Output:

IERROR	Error Flag = 1, Illegal unit number. = 2, Illegal Tree-Type Code. = 3, Illegal Record-Type Code. = 4, Illegal Field Name. = 5, Illegal Code Name. = 6, Internal buffer overflow.

4. SYSTEM RECORDS IN DBMS FILES

In a given DBMS File (System or Data) the System Records mentioned in Chapter 3 contain all the information needed to define the structure of the user Data Records in the File. These are themselves structured in a "Tree" hard-wired in the code of the Basic Package (DBBASP). The File Manager, therefore, is not required to define this particular "Tree". There are 6 different kinds of Record-Types envisaged. Each takes the same "Major" Name – 'FSYSTEM'. The Record-Types and the "Minor" Names are:

1. FILE :
   'FILE' – containing all the information, given at first declaration to the DataBase, about the File,

2. TREE :
   'TREE$Tree-Type-Code' – contains information about a given Tree-Type in the File,

3. RECO :
   'TREE$Tree-Type-Code.RECO$Record-Type-Code'
   – contains information about a given Record-Type in a given Tree,

4. FILD :
   'TREE$Tree-Type-Code.RECO$Record-Type-Code.FILD$Field-Name' – contains information about a given Field in a Record-Type,

5. CODE :
   'TREE$Tree-Type-Code.RECO$Record-Type-Code.FILD$Field-Name.CODE$Code-Name' – contains information about a particular value of a Code for a given Coded Field, and

6. WORD :
   'TREE$Tree-Type-Code.RECO$Record-Type-Code.FILD$Field-Name.CODE$Code-Name.WORD$Word-Number' – contains information relating to a given word in Coded or Fixed Length Field.
   (Note: a Fixed or Variable Length Field is treated by the DBMS as a Coded Field with only a single value for the Code = Field-Name.)

4.1 Record Structure of the System Records

Similar to the Data Records each type of System Record starts with a Variable Length Field – the Header. The format and structure of other Fields in these System Record depends on the type of Record.

1. FILE Record : This record, one per File, has the following Fields :

Field #	Type	Contents
1	F	File Size (2 REAL*4 Words)
2	V	Field Title (Complete 4-byte Words with trailing blanks in the last word, if necessary.)
3	V	Password (complete 4-byte words).
4	V	List of Tree-Type-Codes.

2. TREE Record : One for each different Tree-Type, has the following Fields:

Field #	Type	Contents
1	F	Maximum Set no. of existing Data-Records in any Tree of this Type (1 REAL*4 Word)
2	V	Tree-Type Title (Complete 4-byte Words)
3	V	List of Record-Type-Codes for Records belonging to this Tree-Type.
4	V	List of Root-Names used in Root Data-Records of this Tree-Type.
5	V	List of Root Record-Types of actual Root Data-Records.
6	V	List of Maximum Set #s for each Root.

3. RECO Record : One for each unique Record-Type in a given Tree.

Field #	Type	Contents
1	V	Record Title (Complete 4-byte Words)
2	V	List of possible Parent Record-Types
3	V	List of possible Daughter Record-Types
4	V	List of Field Names

4. FILD Record : One per defined Field in a given Record-Type in a specified Tree-Type

Field #	Type	Contents
1	F	Field Type (Alphanumeric) & Field Number (REAL*4)
2	V	Field Title (Complete 4-byte Words)
3	V	List of Codes (for a Coded Field) Note: For a F/V-type Field only 1 long (=Field-Name)

5. CODE Record : One for each possible Code Value for a Coded Field in a given Record-Type. Also one such Record for a Fixed or Variable Length Field

Field #	Type	Contents
1	F	Field Length (1 REAL*4 Word)
2	V	Code Title (Complete 4-byte Words)
3	V	List of Data Formats for each and every word in the Field. (For a V-type Field the format is the same for each & every word)

6. WORD Record : One per word in defined F- or C-Type Field in a given Record in a Tree, but only one such record (for the first word) for a V-type Field since all words have the same description.

Field #	Type	Contents
1	F	Range – minimum & maximum values (2 Words)
2	V	List of allowed Discrete Values.
3	V	Textual Description of the Word.

It should be noted that for the basic DataBase activity of storage and access, the last type of System Records (WORD) are not essential. They are included in this DBMS to allow for meaningful presentation (textual) and for ensuring valid contents if required in a particular type of application (e.g. storage of Menus & Dialogues).

5. DATA-RECORDS IN DATA FILES

In any given Data File the user Data will be stored in "Basic" Records of the type(s) already defined. For each Basic Record CARGO will permit entering one or more "Update" Records containing the changed values of only a set of the constants in the Basic Record. The Local File Manager or a casual user does not need to define these Update Records. The structure of these Records is pre-fixed in the DBMS.

As stated above each Record – Basic or Update – is made up of Fields of which the first one is the Header Field. An Update Record is made up of one other Variable Length Field beside the Header Field. This field contains, for every word different in value from before, three words – the Field number, the location within the Field and the new value – in this Field. The contents of the Header Field – a Variable Length Field of REAL*4 words – are:

Word #	Description
1	Date of Entry in YYMMDD.
2	Time of Entry in HHMMSS.
3	No. of User-defined Fields
4	Location of 1st User Field
5	Location of 2nd User Field
.	
.	

The user is not normally presented with the contents of this field when accessing a given data record. If required, this field can be accessed by a call to routine DBGETF (see Section 5.3.5).

5.1 Record Naming Convention

KAPACK reads and writes Data records by a name made up of two parts – a "Major" and a "Minor" Name. The naming convention used is as follows:

1. Major Name : 'Tree-Type Code'.SET$xxxx – where xxxx is a version/set number assuming that many versions of the same Data will eventually need to be stored.

2. Minor Name : made up of a series of Record-Type Code and Element name separted by a '$' – each combination separated by a '.' . To distinguish between Basic and Update Records a 2-character postfix is added to this name – viz: '.B' and '.U' . The Minor Name is terminated by a "Period of Validity" – made-up of a start Date & Time and an end Date & Time (in all 25 characters including a leading '.'). The complete minor name therefore reflects the complete Tree Structure to which the Record belongs down to its level in the Tree and indicates the Period of Validity.

   In certain types of applications it may not be desirable to open a new set of Basic + Update records at certain times e.g.

   a. during data-taking on-line where pointers to a single existing set have already been established. So any major alterations have to be entered in an Update Record.

b. only change needed is the increase in the length of a Variable Length Field after data has already been entered.

For such cases the DBMS allows entering a "Special Update" Record belonging to the same set of Basic + Update Records. This record is in fact a complete basic record and is signalled by the postfix '.S' to the minor name.

Below are two examples of minor names of two Basic records belonging to a Tree of three types of Records – AAAA, BBBB & CCCC with BBBB having an AAAA as Parent and CCCC having an AAAA or a BBBB as Parent –

'AAAA$Name_of_AAAA.BBBB$Name_of_BBBB.CCCC$Name_of_CCCC.B.P_of_V'

'AAAA$Name_of_AAAA.CCCC$Name_of_CCCC.B.P_of_V'

5.2 Input of Data-Records

Input of Data-Records is foreseen via the interactive Control, Enquiry and Modification Program (see next Chapter). This can be done interactively with the local Data File Manager entering values of constants on prompts from the program. However, it is expected that many users will contribute data to be input to a particular Data File. Such a situation is extremely laborious if only the Local Data File Manager is expected to interactively enter this data and rather difficult to control if each and every user is allowed to enter the data. It is therefore necessary to have a well-defined format for transfering data from remote sites (and different machines) for input to the Data File centrally without undue overload on the local Data File Manager.

The selected format is to put data onto ASCII Files in as format-free a manner as possible. This choice has been made for the following reasons:

1. There are already enough Software Packages involved in this DBMS without introducing another one for machine-independent binary I/O.

2. ASCII is now a standard used for transmission of Data Files over networks between different machines.

3. ASCII files can be easily edited, using to-day's powerful text-editors, when errors are discovered.

The format of these ASCII Files is as follows :

1. Line 1: the beginning of a Data-Record identifier :*DBRECORD

2. Line 2: the 4-Character Tree-Type Code followed by the Record
   Identifier (see Record Naming Convention above). The
   end of the Record Identifier is searched for by
   looking for the first non-blank character from the
   right of the line. If this happens to be in column 80,
   then a continuation of it is expected on the next
   line, and so on.

3. Line Next : contains the start date (YYMMDD) and time (HHMMSS)
   – separated by a ',' – for the period of Validity for
   this Data-Record. These are followed by the date
   (YYMMDD) and time (HHMMSS) of entry – separated by a ','.
   If any of these dates and times are 0 then the current

Date and Time will be used.

4. Line Next : a Data-Field identifier :*DBFIELD.
   If *DBFIELD is followed by a ',' then the user is expected to supply the format (a la HYDRA Title Cards) to read in all the constants in the Field given on the following Line(s). Otherwise, it is assumed that constants are given in a format-free manner, separated one from the other by a ','.

5. Line Next : the Data words, formatted as indicated.

Note: –

a. It should be remembered that the 1st word is the number of words of data following in the Field and has to be given.

b. Data for **each** field – even if empty – is given, in the order the fields were defined when the Data Structure was entered for the Data Record, in this way until the last Field in the Record.

c. Input for a Data Record is terminated by a new beginning of Record identifier (*DBRECORD) for the next record or by an End-of-File marker if the record is the last one.

The routine used to read-in Data-Records from such ASCII file(s) and add them to the appropriate Data File is DBARIF described below:

5.2.1 DBARIF – Input Data-Records from ASCII File

```
CALL DBARIF (LUN,LUNIN,TRCODE,RTCODE,IERROR)
```

Routine to read Data-Records from ASCII input File and add them to the appropriate Data File after making the appropriate checks.

Input:

LUN	Logical Unit No. of the Data File.
LUNIN	Logical Unit No. of ASCII Input.
TRCODE	Tree-Type Code (CHARACTER*4)
	= '****', all Tree-Types.
RTCODE	Record-Type Code (CHARACTER*4)
	= '****', all Record-Types.

Output:

IERROR	Error Flag

5.3 Data-Record Access Routines

The DBMS allows access to user Data at the complete Record level or at the level of a single defined Field within the Record. For the purpose of storage of user Data the functionality is defined only at the complete Record level (Although in the Control, Enquiry and Modification Package – see later – it may be possible to receive user Data Field-by-Field and build-up the complete Record before writing into the Data File)

The set of functions and the routines to perform them are:

1. DBADDR: to add a Record to a given Data File.
2. DBREPR: to replace an existing Record in a given Data File.
3. DBDELR: to delete an existing Record in a given Data File.
4. DBGETR: to fetch an existing Record in a given Data File. **Note:** Access is controlled by two times – validity & entry. The Data Record fetched satisfies the required time of validity and had been entered before the given time of entry.
5. DBGETF: to fetch a named Field of an existing Record in a given Data File. It should be pointed out that fetching an individual Data Field is rather more costly in access time than fetching a complete Data Record.

5.3.1 DBADDR – Data Storage & Access

```
CALL DBADDR (LUN,TRCODE,RECID,VDTIM,EDTIM,RECORD,IERROR)
```

Routine to enter a new user Data Record in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
VDTIM	Array with Start Date + Time of Period-of-Validity
EDTIM	Array with Date + Time of Entry. If = 0, current date & time are used.
RECORD	the Array containing the user Data-Record.

Output:

IERROR	Error Flag = 1 Illegal unit number = 2 illegal Tree Type Code = 3 illegal Record Identifier = 4 unknown Record Type = 5 illegal Parent Type = 6 variable length Field with negative WC = 7 value out of range in Variable Length Field = 8 illegal value in Variable Length Field = 9 unknown Code

= 10 Coded or Fixed Length Field with wrong WC
= 11 value out of range in Coded/Fixed Field
= 12 illegal value in Coded/Fixed Field
= 13 illegal word counter in an Update Record
= 14 buffer overflow when adding Header Field
= 15 no root Record for this Date & time
= 16 unknown root name
= 17 buffer overflow when replacing TTR
= 18 buffer overflow when replacing an Abstract R.
= 19 write access not allowed
= 20 the Parent Record does not exist
= 21 the Record already exists
= 22 the Basic Record does not exist
= 23 Validity Period incompatible with Basic Record
= 24 Field number incompatible with Basic Record
= 25 Word number incompatible with Basic Record
= 26 Update Record incompatible with existing U.R.

5.3.2 DBREPR

```
CALL DBREPR (LUN,TRCODE,RECID,VDTIM,EDTIM,RECORD,IERROR)
```

Routine to replace an existing user Data Record in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
VDTIM	Array with Start Date + Time of Period-of-Validity
EDTIM	Array with Date + Time of Entry. If = 0, current date & time are used.
RECORD	the Array containing the user Data-Record.

Output:

IERROR Error Flag
= 1 illegal unit number
= 2 illegal Tree Type Code
= 3 illegal Record Identifier
= 4 TTR does not exist
= 5 no root Record for this Date & time
= 6 unknown root name
= 7 Record not found
= 8 variable length Field with negative WC
= 9 value out of range in Variable Length Field
= 10 illegal value in Variable Length Field
= 11 unknown Code
= 12 Coded or Fixed Length Field with wrong WC
= 13 value out of range in Coded/Fixed Field
= 14 illegal value in Coded/Fixed Field
= 15 illegal word counter in an Update Record

= 16 buffer overflow when adding Header Field
= 17 Field number incompatible with Basic Record
= 18 Word number incompatible with Basic Record
= 19 Write access not allowed
= 20 New Record incompatible with existing Update Records

5.3.3 DBDELR

```
CALL DBDELR (LUN,TRCODE,RECID,VDTIM,EDTIM,IERROR)
```

Routine to delete an existing user Data Record in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
VDTIM	Array with Start Date + Time of Period-of-Validity
EDTIM	Array with Date + Time of Entry.

Output:

IERROR Error Flag
= 1 illegal unit number
= 2 illegal Tree Type Code
= 3 illegal Record Identifier
= 4 TTR does not exist
= 5 no root Record for this Date & time
= 6 unknown root name
= 7 Record not found
= 8 Write access not allowed
= 9 If the Update Record is removed the remaining Update Records are incompatible with B.R.

5.3.4 DBGETR

```
CALL DBGETR (LUN,TRCODE,RECID,VDTIM,EDTIM,NMAX,NW,RECORD,
             PERIOD,NUP,IERROR)
```

Routine to fetch an existing user Data Record in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.

VDTIM	Array with Date + Time of Period-of-Validity If = (0,0) the current date & time are used.
EDTIM	Array with Date + Time of Entry. If = (0,0) the current date & time are used.
NMAX	expected number of words in the Record.

Output:

NW	the actual number of words in the Record.
RECORD	the Array containing the user Data Record. Note : If NW > NMAX, then the first NMAX words are returned.
PERIOD	Array with the period-of-Validity of the returned Record.
NUP	no. of Update Records upto the requested Date & Time.
IERROR	Error Flag = 1 illegal unit number = 2 illegal Tree Type Code = 3 illegal Record Identifier = 4 unknown Tree Type = 5 no root record for the given date and time = 6 unknown root name = 7 the wanted record does not exist = 8 number of words > NMAX

5.3.5 DBGETF

```
CALL DBGETF (LUN,TRCODE,RECID,VDTIM,EDTIM,FNAM,NMAX,NW,
                    FIELD,PERIOD,NUP,IERROR)
```

Routine to fetch a named Field from an existing user Data Record in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
VDTIM	Array with Date + Time of Period-of-Validity If = (0,0) the current date & time are used.
EDTIM	Array with Date + Time of Entry. If = (0,0) the current date & time are used.
FNAM	Character*4 – the Field Name.
NMAX	expected number of words in the Field.

Output:

NW	the actual number of words in the Field.
FIELD	the Array containing the user Data Field. Note : If NW > NMAX, then the first NMAX words are returned.
PERIOD	Array with the period-of-Validity of the returned Record.

NUP	no. of Update Records upto the requested Date & Time.
IERROR	Error Flag = 1 illegal unit number = 2 illegal Tree Type Code = 3 illegal Record Identifier = 4 illegal Field Name = 5 unknown Field = 6 no root record for the given date and time = 7 unknown root name = 8 the wanted record does not exist = 9 number of words > NMAX

5.4 Data-Record Utility Routines

Besides the routines, described above, for access of Data-Records, CARGO contains the following routines to perform the interrogatory functions on a given Data File:

1. DBPRID: Print (dump to debug printing file) the complete list of Data-Record Identifiers for a given Tree- and Record-Type.

2. DBPRIR: Print the complete set of Data-Records.

3. DBGNAM: Return list of all Data-Record Names (as opposed to the full Identifiers) for all Data-Records of a given Tree- and Record-Type.

4. DBGNUT: Return the number of Update Records associated to a given Basic Data-Record and their corresponding Periods-of-Validity.

5. DBGRNM: Return the list of names of all "Root" Data-Records belonging to a particular Tree- and Record-Type.

6. DBGBNM: Return the list of names of all "Branch" Data-Records attached to a particular Data-Record.

7. DBGTMW: Return the number of Data-Records with a particular Record Identifier and their corresponding Periods-of-Validity.

5.4.1 DBPRID – Data-Records Utilities

```
CALL DBPRID (LUN,TRCODE,RTCODE,ROOTID,VWINDO,EWINDO,IERROR)
```

Routine to print to debug printing file the complete list of Data-Record Identfiers, the Validity Periods and the Dates & Times of entry of records within the supplied time windows. This can be done for all or a selected Tree- and Record-Types in a given Data File. The output goes to the designated debug printing file.

Input:

LUN	Logical Unit No. of the Data File.

	= 0, if all Data Files.
TRCODE	the Tree-Type Code – a 4-Character String.
	= '****', all Tree-Types.
RTCODE	Record-Type Code (CHARACTER*4)
	= '****', all Record-Types.
ROOTID	Identifier (Character String) of "Root" Record
	= '****', for all possible "roots".
VWINDO	Period-of-Validity
	Array with 4 elements.
EWINDO	Range of Date & Times of entry.
	Array with 4 elements.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Illegal Tree-Type Code.
	= 3, Illegal Record-Type Code.
	= 4, Illegal Root Record Identifier.
	= 5, Buffer Overflow.
	= 6, Stack Overflow.

5.4.2 DBPRIR

```
CALL DBPRIR (LUN,TRCODE,RTCODE,ROOTID,VWINDO,EWINDO,IERROR)
```

Routine to print the complete set of Data-Records within the supplied time windows. This can be done for all or a selected Tree- and Record-Types in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
	= 0, if all Data Files.
TRCODE	the Tree-Type Code – a 4-Character String.
	= '****', all Tree-Types.
RTCODE	Record-Type Code (CHARACTER*4)
	= '****', all Record-Types.
ROOTID	Identifier (Character String) of "Root" Record
	= '****', for all possible "roots".
VWINDO	Period-of-Validity
	Array with 4 elements.
EWINDO	Range of Date & Times of entry.
	Array with 4 elements.

Output:

IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Illegal Tree-Type Code.
	= 3, Illegal Record-Type Code.
	= 4, Illegal Root Record Identifier.
	= 5, Buffer Overflow.
	= 6, Stack Overflow.

5.4.3 DBGNAM

```
CALL DBGNAM (LUN,TRCODE,RTCODE,NMAX,NNAM,NAMES,IERROR)
```

Routine to return the complete set of Data-Records Names ("Root" and "Branch" Data-Records) for a given Tree- and Record-Type in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	Record-Type Code (CHARACTER*4)
NMAX	Maximum no. of Record Names expected.

Output:

NNAM	Actual No. of Record Names returned
NAMES	Character*4 (NMAX) array with Record Names.
IERROR	Error Flag = 1, Illegal unit number. = 2, Illegal or unknown Tree-Type Code. = 3, NNAM > NMAX

5.4.4 DBGNUT

```
CALL DBGNUT (LUN,TRCODE,RECID,DATIM,NMAX,NUP,PEROV,IERROR)
```

Routine to return the number of Update Records, their corresponding Periods-of-Validity and their Dates & Times of Entry, associated to a given Basic Record belonging to a particular Tree-Type .

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
DATIM	Array with Date + Time of validity to identify appropriate basic record.
NMAX	Maximum no. of Update Records expected.

Output:

NUP	Actual No. of Update Records.
PEROV	Array with Periods-of-Validity and Dates & Times of Entry. 6 elements per Update Record. If NUP > NMAX, 1st NMAX periods returned.
IERROR	Error Flag

= 1, Illegal unit number.
= 2, Illegal Tree-Type Code.
= 3, Illegal Record Identifier
= 4, Unknown Tree-Type.
= 5, No "Root" Record for give Data & Time.
= 6, Unknown "Root" Name.
= 7, Specified Data-Record does not exist.
= 8, NUP > NMAX

5.4.5 DBGRNM

```
CALL DBGRNM (LUN,TRCODE,RTCODE,NMAX,NNAM,NAMES,IERROR)
```

Routine to return the complete set of "Root" Data-Records Names for a given Tree- and Record-Type in a given Data File.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RTCODE	Record-Type Code (CHARACTER*4)
NMAX	Maximum no. of "Root" Record Names expected.

Output:

NNAM	Actual No. of "Root" Record Names.
NAMES	Character*4 (NMAX) array with Record Names. If NNAM > NMAX, 1st NMAX names are returned.
IERROR	Error Flag = 1, Illegal unit number. = 2, Illegal or unknown Tree-Type Code. = 3, NNAM > NMAX

5.4.6 DBGBNM

```
CALL DBGBNM (LUN,TRCODE,RECID,DATIM,BRCODE,NMAX,NBR,
                     BRNAMS,IERROR)
```

Routine to return the complete set of "Branch" Data-Records Names of a specified "Branch" Record-Type for a paricular Data-Record of a given Tree-Type.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
DATIM	Array with Date + Time of validity to identify appropriate

	basic record.
BRCODE	"Branch" Record-Type Code (CHARACTER*4)
NMAX	Maximum no. of Record Names expected.

Output:

NBR	Actual No. of Record Names returned
BRNAME	Character*4 (NMAX) array with Record Names.
IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Illegal Tree-Type Code.
	= 3, Illegal Record Identifier
	= 4, Unknown Tree-Type.
	= 5, No "Root" Record for give Data & Time.
	= 6, Unknown "Root" Name.
	= 7, Specified Data-Record does not exist.
	= 8, NBR > NMAX

5.4.7 DBGTMW

```
CALL DBGTMW (LUN,TRCODE,RECID,NMAX,NTW,PEROV,IERROR)
```

Routine to return the complete set of Periods-of-Validity and Dates & Times of entry for a particular Data-Record of a given Tree-Type.

Input:

LUN	Logical Unit No. of the Data File.
TRCODE	the Tree-Type Code – a 4-Character String.
RECID	the complete Record Identifier.
NMAX	Maximum no. of Periods-of-Validity expected.

Output:

NTW	Actual No. of Periods-of-Validity returned.
PEROV	Array with Periods-of-Validity
	and Dates & times of entry.
	6 elements per Record.
	If NTW > NMAX, 1st NMAX periods returned.
IERROR	Error Flag
	= 1, Illegal unit number.
	= 2, Unknown Tree-Type Code.
	= 3, Illegal Record Identifier
	= 4, NTW > NMAX

6. IMPORT/EXPORT OF DATA FILES

An extremely important feature is the ability to import/export between geographically separated sites (and different types of machines)

1. the defined Data-Structure and

2. the complete set of Data-Records

for any/all Data File(s) in the DataBase.

As indicated in Chapter 4 the means selected for such transports is ASCII files over networks (Wide/Local Area).

6.1 Export of a Data File

To export a Data File to a remote site an ASCII file containing the Data-Structure System Records (see Chapter 5) and the Data-Records has to be prepared and transferred into the appropriate place in the remote machine. The steps to be followed are:

1. Write out the complete Data-Structure Records to the ASCII file – Routine DBWRDS.

2. Write out the complete set of Data-Records to the ASCII file – Routine DBWRDR.

3. Transfer the ASCII file using the standard File Transfer Program (FTP) to the remote machine.

6.1.1 DBWRDS – Data File Transport Routines

```
CALL DBWRDS (LUN,LUNOUT,IERROR)
```

Routine to dump the complete set of Data-Structure defining System Records for a given Data File onto an ASCII file in a standard defined format.

Input:

LUN	Logical Unit No. of the Data File.
LUNOUT	Logical Unit No. of the ASCII File.

Output:

IERROR	Error Flag = 1, Illegal Data File unit number. = 2, Illegal ASCII File unit number. = 3, Error writing output File.

6.1.2 DBWRDR

```
CALL DBWRDR (LUN,LUNOUT,TRCODE,RTCODE,ROOTID,VWINDO,
                    EWINDO,IERROR)
```

Routine to dump a given set of Data-Records for a given Data File onto an ASCII file in a standard defined format. The Data Records can be selected by Period-of-Validity and/or over a range of Dates & Times of Entry.

Input:

LUN	Logical Unit No. of the Data File.
LUNOUT	Logical Unit No. of the ASCII File.
TRCODE	the Tree-Type Code – a 4-Character String. = '****', all Tree-Types.
RTCODE	Record-Type Code (CHARACTER*4) = '****', all Record-Types.
ROOTID	Identifier (Character String) of "Root" Record = '****', for all possible "roots".
VWINDO	Period-of-Validity Array with 4 elements.
EWINDO	Range of Date & Times of entry. Array with 4 elements.

Output:

IERROR	Error Flag = 1, Illegal Data File unit number. = 2, Illegal ASCII File unit number. = 3, Error writing output File. = 4, Stack overflow.

6.2 Import of a Data File

Once an ASCII file copy of a DataBase File has been received, the following steps have to be taken to establish the new Data File in the local DataBase

1. declare the new Data File to the local DataBase System File. This is done by routine DBADDF (see section 2.2.1),

2. enter the Data-Structure System Records, reading them off the ASCII input file, to this Data File – Routine DBRDDS

3. write in the complete set of Data-Records from the input file – Routine DBARIF (see section 4.2.1).

6.2.1 DBRDDS – Import Routine

```
CALL DBRDDS (LUNIN,LUN,IERROR)
```

Routine to read the complete set of Data-Structure defining System Records from a given ASCII File and write them onto a new Data File in a local DataBase.

Input:

LUNIN	Logical Unit No. of the ASCII File.
LUN	Logical Unit No. of the Data File.

Output:

IERROR	Error Flag
	= 1, Illegal ASCII File unit number.
	= 2, Illegal Data File unit number.
	= 3, Data File is not empty.
	= 4, Write access not allowed to output file.
	= 5, Error reading input File.
	= 6, Input File is truncated.
	= 7, Buffer Overflow.

The steps, described above, have already been programmed in as selectable options in the DataBase Control, Enquiry and Modification Program (see chapter 7) for both export and import.

7. CONTROL, ENQUIRY & MODIFICATION PACKAGE (DBCEMP)

The routines described in the preceeding chapters form part of the Basic Package (DBBASP) of CARGO. These provide any higher level user program an interface to KAPACK. An interactive Control, Enquiry and Modification Package is also needed at the purely DataBase activity level, as opposed to the application level. Such a Package (DBCEMP) is provided to perform the following functions :

1. create a Structured Data Base from scratch, adding in new Data Files and defining completely the structure of the Data intended to go into these Files,

2. add, modify, delete and retrieve the defined Data-Structure for any given Data File in the DataBase, and

3. add, modify, delete and retrieve data from any given Data File in the DataBase

The program is fully interactive, interaction being via a tree-structured set of menus and within each option, selected from a given branch-menu, all necessary input to/output from the user being done by a well-defined dialogue. This is achieved by the use of the menu and dialogue packages MNPACK and DLPACK (see [1]). NORD, IBM and VAX versions of these packages are now available.

However, also available is a version using the Screen Management Utility routines (SMG) for VAX/VMS Systems. Menus and Dialogue are presented via discrete windows on VT100 equivalent terminals.

DBCEMP is essentially for the use of the 'Managers', individuals responsible for individual Data Files on the DataBase. However, since interrogation of both the defined Data-Structure and the Data Record present on any given Data File is a key function of the program, it can also be more widely used. Protection of both the Structure and the Data is automatically built-in since a user who has not been able to supply the correct local Password is only given Read-only access. Also export and import of the DataBase Files (from and to CERN) can be affected using this program. portability for the moment.

7.1 DBCEMP Operation

The main program requests the logical unit number associated to the System File. Having successfully established that the File assigned to this unit has really the characteristics of a System File, it requests the overall DataBase (System) Password. If the correct one is offered the full access to all Data Files is ensured and the user is presented the options indicated in Menu # 1 – Table 1 below.

Table 1: DBCEMP – Top Menu.

```
ID= PROC= The DELPHI DataBase. Test 12.


A End of Run ?
B Add a New Data File ?
C Delete a File ????
D Activate a File ?
E Export a File ?
F Import a File ?









ALSO VALID 1=TOP 2=BACK
```

If the user is unable to give the correct DataBase Password, or he has selected either to delete or activate a Data File, he is presented the list of Titles of all the Data Files recognised by the DataBase (an example of this is shown below as Menu # 2 – Table 2) and is asked to select one of the Data Files.

On selecting option F the program first attempts to add a new File (option B), getting the necessary File parameters from the user. It then requests the ASCII input File Name and enters in the imported Data-Structure System Records followed by the imported Data-Records.

Option E, on the other hand, results in dumping, in standard format, the System and Data-Records onto an ASCII output File which can then be transferred to any remote machine. Needless to say, the Data File on the local DataBase is not affected.

It should be noted that the program operates on one Data File at a time. Work on more than one file in a given session can only be done sequentially.

Table 2: Menu # 2 – List of Data File Titles.

```
ID= PROC= The DELPHI DataBase. Test 12.
*****    Select a File to ACTIVATE !    *****

A End of Run ?
B Directory File
C Sensing Device File
D Material Constants File.
E The Geometry File.




ALSO VALID 1=TOP 2=BACK
```

Once a particular Data File is selected the program checks if full access to this file has already been obtained (by a correct overall DataBase Password). If not the user is requested for the local Password for this Data File. To be able to

1. define, extend or modify the Data-Structure of Data-Records in this File, and

2. add, delete or modify Data-Records in the Data File

the user has to supply the correct local Password. He can then run down the full series of Menus (Nos. 3 onwards – Tables 3, 4, 5, 6, 7, 8, 9, 10 and 11) shown below.

When the user cannot give the correct local Password he is only offered Read-access to the Data File and is allowed to select the Tree-Type, Record-Type and Field(s) within that type of Record for reading in the appropriate Data. This is done via Menus 5, 7 and 9 (Tables 5, 7 and 9).

At each stage the user has the possibility to go right back to the TOP Menu or the previous Menu provided by the standard bottom 'Return Line' given by MNPACK. However, since it is a long way down from the TOP Menu to Menu No. 11, at each point the user is allowed to jump back to any preceeding level by selecting any of the offered options from 'K' onwards.

Table 3: Menu # 3 – Actions on the Selected Data File

```
ID=  PROC=  The Geometry File.
*****    ACTIONS !    *****

A End of Run ?
B Alter File Parameters ?
C Alter Data-Structure Definitions ?
D Access (Read/Write) Data ?








ALSO VALID 1=TOP 2=BACK
```

Apart form the selection of option A, which is obvious, the following activity is initiated by the choice of any of the other 3 options:

1. Option B implies a change of one or more of the following File Parameters:

    a. the print unit for messages related to this File,

    b. the Data File Title,

    c. the local Password,

    d. the File Size.

2. Option C results in the Menu # 4 (table 4 below) being presented to select the required action at the Tree Level.

3. Option D forces the user to select the Tree-Type (Table 5), followed by the Record-Type (Table 7) and then the Field within that Record-Type (Table 9).

Table 4: Menu # 4 – Actions at the Tree Level

```
ID= PROC= The Geometry File.
*****   Tree-TYPE ACTIONS   *****

A End of Run ?
B Add a new Tree-Type ?
C Delete a Tree-Type ?
D Modify a Tree-Type ?




K Back to list of Data Files




ALSO VALID 1=TOP 2=BACK
```

The choices offered for different types of actions at the Tree level are self-explanatory. Option B engages the user in a dialogue to obtain all the necessary Tree-Type parameters for the new Tree-Type to be added to the Data Structure for the selected File. On the other hand, for C & D the user is presented, Table 5 below is an example, the list of Tree-Types to select the required one.

Table 5: List of Tree-Type Codes.

```
          The Geometry File
--------------------------------------
     1. GEOM    2. MATE
--------------------------------------

Tree-Type to MODIFY/DELETE/ACCESS (END to Finish) ? [HELP] :
```

On requesting "HELP" the list is again presented to the user, this time with the textual description present on the DataBase File for each Tree-Type.

If deleting a Tree-Type this is attempted after confirmation that the user really does want to delete. As stated earlier no changes to the Data Structure are allowed if Data Records pertaining to that bit of the structure are already present. When modifying a Tree-Type, then the various modify options are presented to the user in th form of Menu # 5 (Table 6).

Table 6: Menu # 5 – Actions on selected Tree-Type

```
ID= PROC= Tree of Records containig Geom Constants.
****   ACTIONS on Tree-Type   ***

A End of Run ?
B Alter Tree-Type Parameters ?
C Add a new Record-Type ?
D Delete a Record-Type ?
E Modify a Record-Type ?




K Back to list of Data Files
L Back to File Actions Menu
M Back to Tree Level – Top




ALSO VALID 1=TOP 2=BACK
```

Alteration to any of the Tree-Type parameters is done with the new values being given interactively.

As at the Tree Level adding a new Record-Type to a selected Tree-Type is done by getting from the user all the necessary information. For the other two choices the user is asked to select the appropriate Record-Type from the list (Table 7) of Record-Types belonging to the specific Tree-Type.

Table 7: List of Record-Type Codes.

Tree of Records containing Geometry Constants
1. DETG 2. SDVG

Record-Type to MODIFY/DELETE/ACCESS (END to Finish) ? [HELP] :

On requesting "HELP" the list is again presented to the user, this time with the textual description present on the DataBase File for each Record-Type.

Again a Record-Type is removed from the Data-Structure, if possible, after confirmation that the user really wants to do that. If modifying a Record-Type then the necessary action has to be selected from Menu # 6 – Table 8.

Table 8: Menu # 6 – Actions on Selected Record-Type

```
ID= PROC= Module Record.
****   ACTIONS on Record-Type   ****

A End of Run ?
B Alter Record-Type Parameters ?
C Add a new Field-Type ?
D Delete an existing Field ?
E Modify an existing Field ?




K Back to list of Data Files
L Back to File Actions Menu
M Back to Tree Level – Top
N Back to list of Tree-Types
O Back to Record Level – Top


ALSO VALID 1=TOP 2=BACK
```

The Record-Type parameters are changed as required via a dialogue with the user. For adding a new Field to a given Record-Type after all the necessary information : Name, Title, Type, Format, Word Descriptions etc. is obtained from the user by appropriate prompts. Choice of options D & E results in the presentation of the list of Fields for the selected Record-Type – Table 9 below.

Table 9: List of Field Names.

DETG Record containing Geometry Constants				
1. MRTR	2. SHAP	3. SDIV	4. SEND	5. NABO

Field Name (ALL for complete Record) ? [HELP] :

On requesting ″HELP″ the list is again presented to the user, this time with the textual description present on the DataBase File for each Field.

Again, a selected Field is deleted only after confirmation and if no Data-Records are present. In the modifying mode the user is presented the list options to choose from in Menu # 7 – Table 10.

Table 10: Menu # 7 – Actions on Selected Field

```
ID= PROC= Geometry Shape.
***** Field Parameters to CHANGE *****

A End of Run ?
B Field Title ?
C Field Name ?
D Modify a word description ?
E Add a word ?
F Delete a word ?
G Change Code Parameters ?



K Back to list of Data Files
L Back to File Actions Menu
M Back to Tree Level – Top
N Back to list of Tree-Types
O Back to Record Level – Top
P Back to list of Record-Types
Q Back to Field Level – Top

ALSO VALID 1=TOP 2=BACK
```

The selections and the resulting actions are self-evident.

When accessing data (read/write) – by selection of option D in Menu # 3 above – the user, after he has selected the Tree- & Record-Types, is asked for his choice of Data activity from the options below – Menu # 8 in Table 11. In each of these cases a Data Record can only be fetched from the Data File concerned if the complete Record Identifier (see Glossary) is known. To this aim the user is interrogated, with the help of the known Data-Structure, to build-up the Record Identifier interactively.

Table 11: Menu # 8 – Data Access Options

```
ID= PROC= Module Record.
*****   Data Options   ****

A End of Run ?
B Add a Data Record ?
C Delete a Data Record ?
D Modify a Data Record ?
E Get Data ?




K Back to list of Data Files
L Back to File Actions Menu

N Back to list of Tree-Types



ALSO VALID 1=TOP 2=BACK
```

The package DBCEMP also contains routines to present to the user both the structure Table for a particular Field within a Record or for the complete Record-Type. An example structure of a fixed-length field is shown in Table 12. FMT is the format of each word, LIMS = ON or OFF signifies whether the word was defined to have values within a range or not and 'Discs' = Y or N indicates whether there is a set of declared discrete values that the word is allowed to take. Finally the textual description, as stored in the appropriate System Record in the Data File, of each word is shown.

The tabular presentation of the structural description of a particular Record-Type is made-up from the presentation of the individual Fields within it – as described above.

Table 12: A Typical Structure of a Fixed-Length Field

Field Title : Geometry Shape.

Word	FMT	LIMS	Discs	Description
1	A	OFF	N	A new Code.
2	R	ON	N	Minimum Azimuth.
3	R	OFF	N	Maximum Azimuth.
4	R	OFF	N	Minimum Radius.
5	R	OFF	N	Maximum Radius
6	R	OFF	N	Minimum Z.
7	R	OFF	N	Maximum Z.

REFERENCES

[1] R. Matthews – CERN Computer Program Library Write-up – Z303

[2] G.P.Gopal – DELPHI Detector Description Manual – DELPHI 86–27 PROG–45

[3] W. Bozzoli Malerba. CERN VAXLIB Library Manual, p.18.

APPENDIX 1

GLOSSARY

1. **System File**

   Direct Access File containing relevant information for each of the Data Files in the DataBase.

2. **DataBase Password**

   Character string of any length whose knowledge allows write access to the System File and to all Data Files.

3. **DataBase Title**

   Character string of any length describing the DataBase.

4. **Data File**

   Direct Access File used to store user data.

5. **Local Password**

   Character string of any length whose knowledge allows write access to the Data File.

6. **Data File Title**

   Character string of any length describing the Data File.

7. **User Data**

   Data Words defined by the user and structured in Trees of Records containing Fields of fixed or variable length.

8. **Tree**

   Structure made of Records linked by Parent Branch-Daughter Branch relations. A Record has one Parent (a ″Root″ or a ″Branch″) and many ″Offshoots″. The Root Record has no Parents.

9. **Tree-Type**

   Tree attribute indicating a type of Data-Structure identified by a Tree-Type Code. Trees of a given Type are made of Records of given Types linked by defined Parent Branch-Daughter Branch relations.

10. **Tree-Type Title**

    Character string of any length describing a Tree-Type.

11. **Tree-Type Code**

    String of 4 characters identifying a Tree-Type.

12. **Record**

Set of data words forming a coherent unit of data which can be stored, replaced or deleted. A record has a Validity Period and consists of several Fields.

13. **Record-Type**

Record attribute indicating a type of data-unit identified by a Record-Type Code. Records of a given Type consist of the same number of named Fields.

14. **Record-Type Title**

Character string of any length describing a Record-Type.

15. **Record-Type Code**

String of 4 characters identifying a Record-Type.

16. **Field**

Set of contiguous words, within a Record, forming the lowest coherent unit for access purposes.

17. **Field Title**

Character string of any length describing a Field.

18. **Field Name**

String of 4 characters which identifies a Field within a Record.

19. **Field Number**

Sequential Number of a Field within a Record.

20. **Fixed Length Field**

Field whose length is the same in all the Records of a given Type.

21. **Variable Length Field**

Field whose length can be different in different Records of the same Type.

22. **Coded Field**

Field whose length depends on a parameter (Code) stored in the Field itself. The Code has a finite number of definite values. Consequently the length of such a Field can be different in different Records of the same Type.

23. **Header Field**

The first Field in a Record with Field Name 'HEAD' and Field Number 0. The Header Field is a Fixed Length Field whose format is imposed by the DBMS.

24. **Word Title**

Character string of any length describing a Data Word in a Field.

25. **Record Name**

String of 4 characters differentiating a Record from other Records of the same Type linked to the same Parent Record.

26. **Record Identifier**

Character string identifying a Record in a Tree. The Identifier of a Record is given by the Identifier of its Parent without its postfix followed by '.Record-Type-Code$Record-Name.X'. The character X at the end of the name indicates whether it is a Basic (X='B'), Update (X='U') or Special Update (X='S') Record.