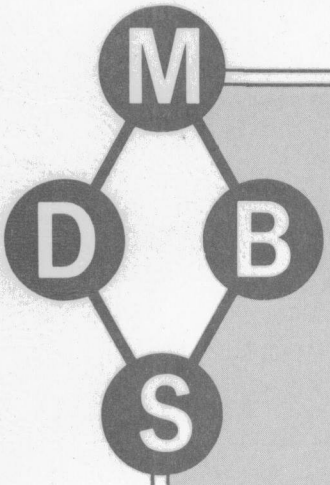


Micro Data Base Systems, inc.
P.O.BOX 248 LAFAYETTE, IN 47902

USER'S MANUAL

MDBS·DMS

MDBS·DDL

MDBS Data Management System Documentation

Version 1.04

Micro Data Base Systems, Inc.

P. O. Box 248

Lafayette, Indiana 47902

(317) 448-1616

(317) 742-7388

August 1980

Copyright Notice

PREFACE

This manual is concerned with data base management and its use as a tool in software development. Many experts have predicted that two innovations will dominate the computing scene in the 1980's. The first innovation is in the area of computer hardware with the development of the micro-computers (computer on a chip). The second, a software advance, is the use of highly sophisticated data base management techniques to control complex data structures. These two innovations are combined together in the Micro Data Base Systems' sophisticated Data Base Management System. We believe that data base management systems (DBMS) with their capability of supporting selective retrieval to "what if" type inquiries and their ability to support restructuring as the nature of the demands change, will become the focal software in micro's as they are already in mini- and maxi-computers. The micro-computer, with its fantastic computing power on a per-dollar basis, combined with a powerful data base management system, will be a formidable tool for managing enterprises of all types.

The purpose of this user's manual is twofold. First there is introductory material to the area of data base management. Since this is such a vast and important area we really can't do complete justice to this topic and suitable references are:

1. Haseman. W.D. and A.B. Whinston, Introduction to Data Management, Richard D. Irwin. Inc., Homewood, IL, 1977.
2. Martin, J., Computer Data-Base Organization, Prentice Hall, Englewood Cliffs, NJ, 1975.

3. Holsapple, C., Primer on Data Base Management, MDBS Inc., Box 248, Lafayette, IN, 1979.

The second goal of this manual is to give a detailed guided tour through the use of MDBS.DDL and MDBS.DMS. Here our aim is to be as complete and as comprehensive as possible.

Finally, we request comments from our readers as to how successful, in their view, we have been in achieving these goals. Have you been able to follow our description of how MDBS.DDL and MDBS.DMS should be used and actually have you achieved the expected results? Or have there been points where the steps were unclear or ambiguous? We plan to incorporate these suggestions in future versions to finally achieve a manual that is a fit companion to what we feel is a remarkable and truly innovative software product in the micro-computer area.

TABLE OF CONTENTS

DDL COMMANDS

(a) By Mnemonic

DDL COMMANDS

(b) By Function

DDL ERROR MESSAGES

INDEX OF DML COMMANDS

NEW RELEASES, VERSIONS, AND A WARNING

Any programming endeavor of the magnitude of the MDBS systems is bound to continue to evolve over time. Realizing this, Micro Data Base Systems vows to provide its users with updates to their version for a nominal handling fee. Updates can be provided only after the signed End User Agreement Form is on file with MDBS, Inc.

New versions of MDBS software will be considered as separate products. However, owners of previous versions will be entitled to a preferential rate structure (as soon as the signed End User Agreement Form is received by MDBS, Inc.).

Finally, each copy of MDBS products is personalized with the licensee's name. There are several levels of this personalization, some of which involve encryption methods guaranteed to be combinatorally hard to decypher. MDBS products were produced with a sizable investment of capital and labor to say nothing of the years of prior involvement in the data base management area by principals of MDBS. Accordingly, we are seriously concerned about any unauthorized copying of our products and will take any and all available legal action against illegal copying or distribution of our products.

I. INTRODUCTION

The Micro Data Base Systems (MDBS) has components for defining data structures and for the storage and retrieval of data. Add-on packages available for MDBS allow the logging of transactions for recovery, the restructuring of a data base, and query handling. MDBS is not a mere file management system. The ideas used in designing MDBS take the CODASYL Data Base Task Group Report (April 71) as a starting point. However, MDBS provides many additional data structuring and data manipulation features that are not available in strictly CODASYL data base systems. MDBS is implemented entirely in machine language.

For each application system a logical structure of the data base must initially be defined; that is, a formal definition is made in which the relationships between the data elements is presented. Initially a user may list the various data items (field names) that will be involved in a particular application. The process of defining a logical data structure consists of grouping data items into record types (or, in conventional data processing terminology, into logical files) and indicating appropriate relationships between record types. These relationships are only conceptual and do not imply any physical storage allocation. Nowhere in the use of the data management system does a user refer to an actual data storage location, but rather, he refers to the conceptual structure as initially defined. This defining of the data structure is done via the DDL (Data Definition Language) and processed by the program named MDBS.DDL.

After the structure has been defined the application programs will, of course, need to access data from or place new data into the data base. An application program does not, however, read (or write) the desired data directly from (or to) the data base. Instead, it requests the data management routines (MDBS.DMS) to perform the necessary operations. This requires the writer of the application programs to know only the conceptual structure of the data base; the DMS is responsible for maintaining the physical structure. The requests to the DMS are made via subroutine calls which make up a collection of commands comprising the DML (Data Manipulation Language).

The basic MDBS package consists of MDBS.DDL and MDBS.DMS. There are other, very useful, add-ons to MDBS. Occasionally, it may become necessary to alter the logical structure of an existing data base. The MDBS.DRS system is designed to permit changes to be made to a data base structure without the need to dump and reload the data base. Another add-on is MDBS.RTL which logs all transactions made on a data base since the last data base backup. In the event of a system crash (e.g., power loss) the data base can be restored automatically by invoking a recovery utility. A third add-on is MDBS.QRS which accepts English-like, nonprocedural queries and produces desired reports. This cuts programming effort substantially. Full details about these add-ons may be found in the MDBS.DRS, MDBS.RTL, and MDBS.QRS User's Manuals.

II. MDBS.DDL

A. Introduction

In this section, we concentrate on Micro Data Base System's Data Description Analyzer/Editor which we call MDBS.DDL (for Micro Data Base System's Data Description Language).

In Part B we list several features of MDBS.DDL which are then described in more detail in later sections. In Section C the user is instructed on how to "bring-up" MDBS.DDL; the kind of modifications that can be made to this are also described.

To develop a data base proper, data base management concepts are necessary. In Section D, we discuss such concepts and present the data description language (DDL) for describing a data base structure. The user may also want to look ahead to Section III.D to see how such concepts are used.

To create a data base, the user must first describe the structure of the data using the data description language (as discussed in Section D) and then physically initialize the data base with the proper tables. This is all done by the DDL analyzer/editor which is discussed in Section E. The analyzer/editor permits the input, editing and analysis of a data description.

B. Features

MDBS.DDL allows the user to describe a data base structure and initialize a data base. A data base structure is defined using the data description language (DDL). Such a description is entered into the computer (either in lower or upper case) via the Text Entry/Command mode of MDBS.DDL. This mode supports:

1. Text entry
2. Listing
3. Deleting
4. Saving
5. Retrieving
6. Renumbering

as well as providing formats and other text entry aids.

Once entered, the text can be edited using the line editor of MDBS.DDL. Finally, the DDL analyzer processes a data base description and, if no errors are detected, initializes a data base. If a syntactical or logical error is detected, an error message is displayed and the user can quickly correct the problem using the text entry or edit features.

MDBS supports the following features for data base design work:

1. Variable and fixed length records.
2. Character, integer, floating point (real), logical, internal decimal, external decimal, and binary record fields (data items).

3. One-to-one, one-to-many, many-to-one, and many-to-many set types.
4. Sorted, LIFO, FIFO, next, prior, and immaterial set orderings.
5. Automatic or manual record insertion into sets.
6. Read and write access protection via passwords at the item, record and set levels of organization.
7. Record types may own other occurrences of the same record type (i.e., recursive sets).
8. Many record types can participate in a given set.
9. Full network data structures.

These features are discussed in Section D.

A data base can be physically spread over a number of drives (8 maximum) and the drives can be floppy (mini-or full- sized) or hard disks.

The data base is organized using a paging system. A single drive can logically contain 8191 pages and a page is logically restricted to, at most, 65536 bytes. Thus large data bases can be supported (subject to operating system constraints on the size of a file).

Once a data base description has been entered and a data base initialized, the user can easily access the data base through a host language (such as BASIC) using MDBS.DMS. In Section III we take up the discussion of MDBS.DMS.

C. Getting Started with MDBS.DDL

1. Initial Loading and Test Run

The details specific to your system for making an initial test of the MDBS.DDL package are outlined in the MDBS system specific manual which is supplied when you purchase the system. Briefly, you will execute program DDL (see the manual MDBS system specific manual for its fully qualified name) which will display:

```
MDBS.DDL VER X.X
(C) COPYRIGHT 1979, 1980, Micro Data Base Systems, Incorporated

Reg # XXXXX
Your name and
address
```

At this point you can read a sample data base description[1] stored in file INVNTRY (again see the system specific manual for the fully qualified name). The procedure is shown below (underlined text is generated by the MDBS.DDL system):

[1] This example was used by M. Gagle, G. Koehler, and A.B. Whinston in their article "Data-Base Systems And Micro-Computers: An Overview", published in BYTE magazine.

R	"Read a File" command
FILENAME	computer prompt
INVNTRY	fully qualified file name for INVNTRY

To list this sample description type "L". To actually initialize a small data base with this description type DDL. This will result in the prompt:

DATA-BASE FILENAME?

to which you should respond with a fully qualified file name which exists on the first drive of your system (use a temporary file for this purpose). The data description will then be displayed (without line numbers) and a data base initialized.

The complete sequence looks like

DDL	"Enter DDL Analyzer" command
DATA-BASE FILENAME?	computer prompt
name	fully qualified file name
(DDL output)	computer generated output
DDL PROCESSING COMPLETED	

If you have not had a successful run, re-read this section and the system specific manual and try again. If you still have no luck, please call Micro Data Base Systems, Inc. for help.

2. Relocating MDBS.DDL

Under some operating systems, it may be desirable to locate MDBS.DDL at a position other than that supplied by Micro Data Base Systems. For this purpose, we have provided a relocatable form of the data description analyzer and a relocator so that an executable form of the analyzer can be ORGed to any place in memory. Please refer to the system specific manual for further information.

3. Important Addresses: Personalizing and Patching MDBS.DDL

MDBS.DDL consists of a program region and work area region. These are contiguous and the work area immediately follows the program area. The size of the work area can be increased or decreased through a patch to the system.

There are several addresses that the user should be aware of in MDBS.DDL. The user may alter these as shown in the system specific manual. A brief description of each item follows.

(a) Initial Entry Point

Upon entry at this point, all program variables and regions are either physically or logically re-initialized. Registers are not saved but the entering stack is preserved.

(b) I/O Entry Points

Different operating systems handle input and output to terminals, printers and disks in a variety of ways. To keep this manual free of system specifics the I/O information relevant for your system is published in the system specific manual. Patches to non-standard I/O routines should be made in accordance with the instructions of the system specific manual.

(c) Echo Toggle (Default 00 hex)

This byte is checked to see if the user wants to have MDBS.DDL echo input to the output device. If the byte value is zero, echoing will take place. If it is the value one, no echoing will be performed.

(d) Last Word of Memory (Default 0BFFF hex)

The address stored here gives the last available word of memory that the MDBS.DDL program may use. Note that MDBS.DDL uses all memory starting from its load address up to the value in this field. Needless to say, the user should make sure that the last word of memory is physically beyond the end of the program.

(e) Screen Control Byte (Default 0B hex)

This byte should have one of the following values:

Screen Width	Byte Value
less than or equal to 64 characters	11 (0B hex)
greater than or equal to 80 characters	15 (0F hex)
64 to 80 characters per line (call it N)	greatest integer less than or equal to: N/5 -1

(f) Re-entry Point

If the user wishes to re-enter MDBS.DDL while preserving all program variables and regions, then he must issue a jump to this address.

In the next section we discuss the data description language. The data description language and features of a data base design are illustrated with examples.

D. DDL (Data Definition Language)

1. Introduction and Definitions

The DDL is used to formally describe a conceptual (or logical) structure of a data base. These descriptions are in terms of data items, record types, and sets.

A DATA-ITEM is the smallest unit of named data. An ITEM-OCCURRENCE is a representation of a value of a data-item. (Eg. AGE is a data-item, and an age of 21 might be an occurrence of AGE) A "repeating item" acts as a one-dimensional array whose maximum replication factor must be specified; a "depending on" item may optionally be specified which indicates the current "length" of the "array."

A RECORD-TYPE is a named collection of zero, one, or more data-items. A RECORD-OCCURRENCE is a sampling of values of the data-items defined to be contained by the record-type. There may be an arbitrary number of occurrences in the data base of each record-type specified. No two record-types may have the same name. The data-item names must be unique only within the record-type. (The same data-item name may be used in more than one record type.) The order, type, and size of the items within a record type are defined in the record description.

Consider the following example of a record description:

```
RECORD EMPLOYEE
ITEM   NUMBER   INT    8
ITEM   NAME     CHAR  20
ITEM   WAGE     REAL   8
ITEM   TAX      REAL   8
```


The above example defines a record-type EMPLOYEE containing four data-items: NUMBER (employee number), NAME, WAGE (wages) and TAX (taxes withheld). The "types" of the data-items are integer, character, real, and real, respectively, and NAME is specified to be a maximum of 20 characters in length. An occurrence of this record-type might be (1520, A B SMITH, 7520.20, 158.42). There would be an occurrence for each employee in the company.

"Record-type" is used in defining a structure, and "record occurrence" is used to refer to the actual data values. A group of data values may make up a record occurrence, and they are stored using the name of the record-type and the names of the data-items. If the names of all employees are desired (using the definition as in the above example), the application program would request of the DMS all occurrences of the record-type EMPLOYEE and for each one it would request the occurrence (the value) of the data-item NAME.

A SET is a named relationship between record-types. One or more record-types are declared as the "owners" and one or more record-types are declared as the "members" of the set. Any record-type may be declared as the owner record-type of one or more sets; likewise for member record-types. A set occurrence may have an arbitrary number of occurrences of each of the record-types. The "set order" (the logical order of the member record occurrences) must be declared.

The set is the basic structural unit of the data base. It is used to define the relationship between different record-types. In particular, the set links each owner record occurrence to its related

member record occurrences. This is best explained through examples.

2. DDL Example

A simple but common business problem is the need to maintain lists of people or organizations sorted in different orders. Consider the case of a company which maintains lists of customers sorted by name, address, and zip code. The MDBS system handles this problem using the data structure of Figure II.D.1. The actual options available for the data definition are detailed in the remaining portions of this section. Note that only one record type (CUSTOMER) has been defined, while three sets (NAMES, NUMBERS and ZIP) are shown. The record type SYSTEM, used as owner in these sets, is a special record predefined by MDBS.DDL which permits access to data in the data base. There is only one instance of the SYSTEM record-type in the data base and there is no data associated with this record-type. The SYSTEM record is discussed later in this section.

The CUSTOMER record type contains all the data relating to a customer. Data-item NUMBER contains the customer number, item NAME the customer name, etc. All of the data-items store character data, as indicated by the CHAR specification. The item size follows the word CHAR.

The use of three sets is the key to maintaining the customer file in three different orders. The MDBS.DMS system will automatically insert a record in the proper location in each of these three sets when the record is created. Note that this is a logical construct only -- a record is only physically present in one place. By use of

appropriate DML commands, a given record can be accessed efficiently, or, by use of other DML commands, a sorted list of records can be accessed. Examples of such programs are illustrated in Section III.D.

All three sets in Figure II.D.1 are SORTED. The "sort key" is stated for each set (NAME for set NAMES, etc.) and is the field defined for the set to be sorted upon. Each customer record occurs only once in the data base, yet three logical set orders exist.

There are six allowable set orders. Besides SORTED, there are also: FIFO - record occurrences are added to the end of the set such that each new record occurrence becomes logically the last member of the set; LIFO - record occurrences are added to the front of the set such that each new record occurrence becomes logically the first member of the set; NEXT - a new record occurrence is logically placed in the set after the "current" member record of the set ("current" is defined in the DML discussion); PRIOR - a new record occurrence is logically placed in the set before the "current" member record of the set; and IMMAT - the user does not care about the set order. Note that the IMMAT set ordering allows the MDBS system to realize certain efficiencies and should be used if possible.

In the set description of Figure II.D.1 the specification "AUTO" appears. This indicates that as occurrences of record type CUSTOMER are created, the records will automatically be inserted in set ONORDER by the MDBS system. The word "MAN" (for manual) could also have been specified. These options are discussed further in Section II.D.5. Also, the specification "1:N" states that the relationship between the

owner of the set and its members is a one-to-many relationship. For each owner occurrence there may be many member occurrences, but not vise versa. Most data base systems support only this type of relationship -- a many-to-many (N:M) relationship supported by the MDBS system is described in the next section.

Developing the DDL for a data base to be used in the MDBS system does not require any knowledge of the DML (Data Management Language). The user needs only to define the data base structure; the system will take care of data storage and retrieval for application programs. However, much insight will be gained if the section describing the DML is read (Section III.D). By seeing what commands are available, it will be easier to understand how to design the data base and establish set relationships. Also, the additional examples will be extremely helpful.

Our example of Figure II.D.1 can be extended to include more information. Suppose that we wish to keep a record of orders that a customer has placed. Conceptually, each customer may have several (or one or even zero) orders that have been placed. Each order though is associated with just one customer. Thus we have what is called a "one-to-many" structure (one customer to many orders), which is a natural structure for a data base set relationship. In order to extend the example of Figure II.D.1, we define a second record type which represents a customer order (see Figure II.D.2). Of course, to actually define the complete DDL for this example, we would add the new record description after the description for the CUSTOMER record.

Note that in record type ORDER we have defined five data items: NUMBER (for order numbers), DATE (order date), PART (the stock number of the part ordered), QUANTITY (the quantity ordered) and PRICE (the invoice price). Three of the data items hold character data, but QUANTITY and PRICE have data types BIN (Binary) and REAL respectively. QUANTITY is stored as a binary data item to reduce storage requirements. This is reasonable since QUANTITY will never exceed 65535 in this application. The binary data type is used to store integral values in binary format instead of the packed decimal format common to many BASICs. The digit "2" after BIN indicates that two bytes should be allocated to store the binary value. The REAL specification for PRICE is a real number whose data length is 8 bytes. LOG (Logical) and INT (Integer) data types are also supported.

The set ONORDER links each customer to his orders. The set ordering is FIFO, so that, for each customer, the orders will be maintained in a first-in, first-out basis, which will normally be the sequence in which the orders were received. The new set description would be added to the DDL after all the record descriptions are given.


```
RECORD CUSTOMER                                          Customer record
ITEM   NUMBER    CHAR  8                                 Customer number
ITEM   NAME      CHAR 20                                 Customer name
ITEM   ADDRESS   CHAR 20                                 Street address
ITEM   CITY      CHAR 20                                 City
ITEM   ZIP-CODE  CHAR  5                                 Zip code

SET    NAMES     AUTO 1:N                                Sorted by Name
                                SORTED   NAME
OWNER  SYSTEM
MEMBER CUSTOMER

SET    NUMBERS   AUTO 1:N                                Sorted by Number
                                SORTED   NUMBER
OWNER  SYSTEM
MEMBER CUSTOMER

SET    ZIP       AUTO 1:N                                Sorted by Zip code
                                SORTED   ZIP-CODE
OWNER  SYSTEM
MEMBER CUSTOMER

END
```

FIGURE II.D.1
DDL Declarations for Multiply Sorted Records


```
RECORD ORDER                                        Customer orders
ITEM   NUMBER    CHAR  6                            Order number
ITEM   DATE      CHAR  8                            Date received
ITEM   PART      CHAR  6                            Part number
ITEM   QUANTITY  BIN   2                            Quantity ordered
ITEM   PRICE     REAL  8                            Unit cost

SET    ONORDER   MAN   1:N                          Link customers
                                 FIFO                to orders
OWNER  CUSTOMER
MEMBER ORDER

END
```

FIGURE II.D.2

DDL Declarations for Customer Orders

3. Many-to-Many Example

Standard (CODASYL) data base systems require a "one-to-many" relationship between the owner and the members of a set occurrence. This restriction may force the use of unnatural data structures, which complicates DML programs unnecessarily, making the program harder to conceptualize and to maintain. These unnatural structures are also quite inefficient in terms of both storage and processing. As an example, imagine a computerized bibliography system in which we have book titles that may be classified by a number of keywords. We wish to be able to obtain: 1) A sorted list of books corresponding to a given keyword, and, 2) A sorted list of keywords corresponding to a given book. Since each keyword describes many books, and since one book can have several keywords, a many-to-many relationship exists between keywords and books.

If we are restricted to standard "one-to-many" sets, we would be forced to introduce artificial "link records" which are used to simulate many-to-many sets in conventional database systems. These link records force us to adopt a highly unnatural, restricted data structure which, since the link records contain no data, has little conceptual value. A DDL (Data Description Language) description of such a structure is shown in Figure II.D.3.

The set relations involved are all one-to-many relationships -- however, we must create an occurrence of record type LINK for each book-keyword pair in our database. In a more complex example (such as when authors are introduced), the actual data relationships present

quickly become unclear. An additional problem with this structure is that it is quite difficult to maintain a sorted order between the books and keywords. Also, the use of link records results in a wastage of data base storage space.

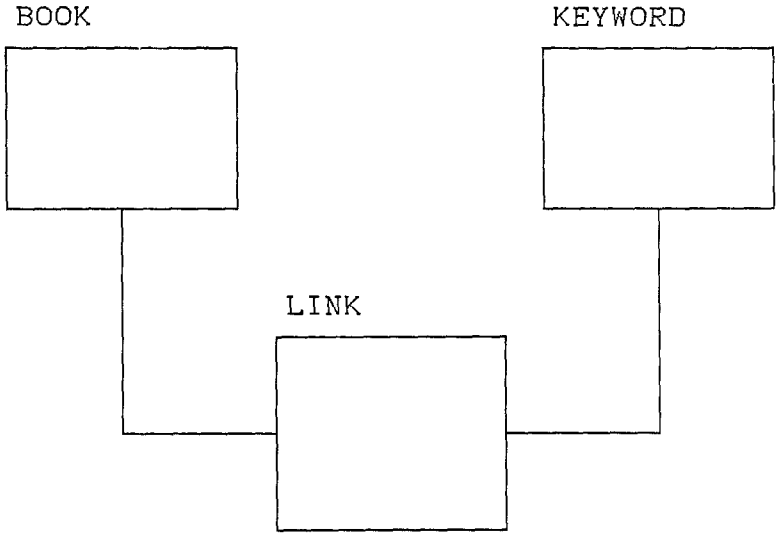
The MDBS Data Management System permits explicit use of many-to-many sets. A special internal structure is used which allows the MDBS system to automatically maintain the data base pointers necessary to represent such sets. This means that situations such as the bibliography example can be conveniently handled by the schema declaration of Figure II.D.4.

In the declaration for set S3, the specification of a second set ordering indicates that the set is a many-to-many set and that the owner records are to be maintained in sorted order. The first set ordering specification indicates that the members of the set are also to be maintained in a sorted order. Note that this feature allows one to either find all keywords for a given book or to list all books for a given keyword.

Use of many-to-many sets blurs the distinction between the owners and members of a set, since the rigid one-to-many ordering is no longer required. The terms "owner" and "member" are thus arbitrarily assigned and are used for notational convenience only.

Many-to-many sets are processed like conventional (one-to-many) sets - i.e., DML commands such as Find Next Member (FNM) and Find Member based on Sort Key (FMSK) are used. Additional DML commands have been defined to process the owners of a set: Find Next Owner

(FNO) and Find Owner based on Sort Key (FOSK). These are fully described in section III.



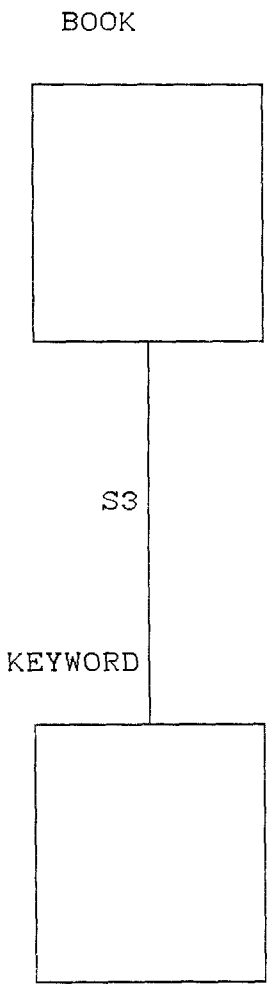
Data Diagram for Many-to-Many Example

FIGURE II.D.3 (a)

```
RECORD BOOK                                          Book record
ITEM   TITLE     CHAR 30                             Book title
ITEM   AUTHORS   CHAR 30                             Author(s)
ITEM   PUBLISHR  CHAR 60                             Publisher

RECORD KEYWORD                                       Keyword
ITEM   KEYWORD   CHAR 10

RECORD LINK                                          Link record


SET    S1        AUTO 1:N                            Sorted list
                                 SORTED   TITLE      of books
OWNER  SYSTEM
MEMBER BOOK


SET    S2        AUTO 1:N                            Sorted list
                                 SORTED   KEYWORD    of keywords
OWNER  SYSTEM
MEMBER KEYWORD


SET    S3        MAN  1:N                            Linkage set
                                 IMMAT               for books
OWNER  BOOK
MEMBER LINK


SET    S4        MAN  1:N                            Linkage set
                                 IMMAT               for keywords
OWNER  KEYWORD
MEMBER LINK

END
```

DDL for Link Record Example

FIGURE II.D.3 (b)



Data Diagram for Many-to-Many Example

FIGURE II.D.4 (a)

```
RECORD BOOK                                              Book record
ITEM   TITLE     CHAR 30                                 Book title
ITEM   AUTHORS   CHAR 30                                 Author(s)
ITEM   PUBLISHR  CHAR 60                                 Publisher

RECORD KEYWORD                                           Keyword
ITEM   KEYWORD   CHAR 10


SET    S1        AUTO 1:N                                Sorted list
                                 SORTED   TITLE          of books
OWNER  SYSTEM
MEMBER BOOK


SET    S2        AUTO 1:N                                Sorted list
                                 SORTED   KEYWORD        of keywords
OWNER  SYSTEM
MEMBER KEYWORD


SET    S3        MAN  N:M                                Book/Keyword
       SORTED    TITLE          SORTED   KEYWORD        relationship
OWNER  BOOK
MEMBER KEYWORD

END
```

DDL for Many-to-Many example
FIGURE II.D.4 (b)

4. Multiple Owner/Member Example

Most set relationships have one record type defined as the owner of the set type and another defined as the member record type. An example of this would be a set (SET1) with owner record type DEPT (Department) and member EMPLOYEE, corresponding to a grouping of employees within departments. Suppose record type EMPLOYEE contains payroll information such as hourly wage, union membership, and seniority. If we wish to define a record type SALARIED corresponding to salaried employees (which presumably would require different payroll information than hourly employees), it might be desirable to allow both record types EMPLOYEE and SALARIED to be possible members of SET1. This is permitted in MDBS; in fact, as many record types as needed may be defined as members (or owners) of a set type. Special MDBS routines are available to determine the actual record type of the active member (or owner) of a set.

This concept is further extended in MDBS by allowing a set to have the same record type as both its owner record type and its member record type. Imagine a set SUPERVIS which indicates the hourly employees (EMPLOYEE) who supervise other hourly employees. A hierarchy of supervision can be conveniently represented by such a data structure (Figure II.D.5).


```
RECORD EMPLOYEE                                          Employee record
ITEM   NUMBER     CHAR  6                               Employee number
ITEM   STATUS     CHAR  6                               Employee status
ITEM   NAME       CHAR 20                               Employee name
ITEM   WAGERATE   REAL  8                               Hourly rate
ITEM   ADDRESS    CHAR 20                               Street address
ITEM   CITY       CHAR 25                               City and state

RECORD SALARIED                                          Salaried  employee
ITEM   NUMBER     CHAR  6                               Employee number
ITEM   STATUS     CHAR  6                               Employee status
ITEM   NAME       CHAR 20                               Employee name
ITEM   WAGERATE   REAL  8                               Hourly rate
ITEM   ADDRESS    CHAR 20                               Street address
ITEM   CITY       CHAR 25                               City and state

SET    S1         AUTO 1:N                              Employee list
                                  SORTED   NUMBER       by employee number
OWNER  SYSTEM
MEMBER EMPLOYEE

SET    S2         AUTO 1:N                              Salaried employees
                                  SORTED   NUMBER       by employee number
OWNER  SYSTEM
MEMBER SALARIED

SET    SUPERVIS   MAN  1:N                              Supervisory
                                  SORTED   NUMBER       relationships
OWNER  EMPLOYEE
OWNER  SALARIED
MEMBER EMPLOYEE
MEMBER SALARIED
```

DDL for Employee Supervision Example

FIGURE II.D.5

Finally, consider a data base used to cross-reference computer subroutines. It is desirable to be able to obtain sorted lists of all subroutines which call a given routine and of all subroutines called by a given routine. Consider record type SUBR in Figure II.D.6.

Record type SUBR stores the name of each subroutine in the system. Set S1 allows any subroutine name to be accessed easily. Set CALL is a many-to-many set type with record type SUBR as both owner and member. To obtain a sorted list of all subroutines that call a specified routine, all that is necessary is to make the specified routine the current member of set type CALL and then access the owner records in sequence. Similarly, a specified routine could be made the current owner of set type CALL and the members of the set occurrence accessed in turn to produce a sorted list of subroutines called.

Much more powerful data structures than those possible with other data base systems can be conveniently used due to MDBS's extreme flexibility. The basic tools are present in MDBS to process even the most complex data structures in a clear, intuitive way.


```
RECORD SUBR                                                  Subroutine name
ITEM   NAME     CHAR  8

SET    S1       AUTO  1:N                                   Sorted list of
                                  SORTED   NAME              names
OWNER  SYSTEM
MEMBER SUBR

SET    CALL     MAN   1:N                                   Subroutine calling
                SORTED NAME  SORTED   NAME                  relationship
OWNER  SUBR
MEMBER SUBR
END
```

DDL for Subroutine Cross-reference Example

FIGURE II.D.6

5. DDL Specification

In the following lists a summary of the parameters for each of the DDL specifications is presented. This section details each of the parameters in detail. The order in which the sections must be presented to the DDL analyzer is:

```
FILES (optional)
    DRIVE
PASSWORDS
RECORD
    ITEM (optional)
SET
    OWNER
    MEMBER
END
```

Each line of text must be preceded by a 4 digit line number and a blank space so that column 1 would refer to the sixth character entered (See the P command of Section II.E). Figure II.D.7 illustrates a typical DDL stream. This Figure appears following descriptions of the various kinds of lines that can appear in a DDL specification stream.

Before presenting the layouts, the following summary is given on the various relationships between access levels one must adher to:

For any data item type, record type or set

its write access level ≧ its read access level

For any data item type

its read access level ≧ the read access level of its record type

its write access level ≧ the write access level of its record type

For any variable length item type

its read access level = the read access level of its depending on item

its write access level = the write access level of its depending on item

For any record type

its read access level ≦ the read access level of any set in which it participates

its write access level ≦ the write access level of any set in which it participates

DDL errors will result if these conventions are not observed.

It is recommended (but not mandatory) that the data base designer observe the following convention:

For any sort key

its read access level ≦ the read access level for the set(s) which it is a sort key

its write access level ≧ the write access level for the set(s) which it is a sort key

No DDL errors will result if a designer decides not to follow this convention.

DRIVE Line

The DRIVE line specifies the number of data base pages that reside on a particular drive. The user's first physical drive is called drive 1. The second drive 2, etc. The DRIVE line layout is:

Columns	Parameter Description
1-5	"DRIVE" - line type
8	Drive number
12-15	Number of pages allowed on

Notes:

1. The drive numbers must be one of 1,2,3,4,5,6,7, or 8.

2. The last drive line will be used if there are two or more drive lines with the same drive number.

3. The number of pages must be non-zero.

4. The DRIVE lines must immediately follow a FILES card.

END Line

The END line is used to signify the end of the data base description to the DDL analyzer. The format is:

Columns	Parameter Description
1-3	"END" - line type

FILES Line

The FILES line and FILES section (which includes the DRIVE lines) are used to define the data base name, size and location. This is an optional section which, if missing from a data description, is supplied by the system using standard defaults.

The FILES section information consists of:

1. The data base filename,

2. The maximum number of disk drives on which the data base will reside (default is 1), and

3. The data base page size (default is 512).

If the FILES section is not present, the user is prompted for a fully qualified filename by:

DATA-BASE FILENAME?

This name, together with the standard defaults, is used in place of the FILES section.

The FILES line format is:

Columns	Parameter Description
1-5	"FILES" - line type
8-19	Data base file name. This must be fully qualified according to the conventions of the operating system.

23	Maximum disk drive number. Calling the first drive number 1, this field must contain the number of the maximal such index.
26-29	Data base page size. This must be a multiple of 256. A recommended value is 512.

Notes:

1. When the FILES section is missing, the defaults are equivalent to:

```
FILES "prompt name" 1 512
DRIVE 1 50
```

2. Suppose the user wanted his data base on the first and third drive. Column 23 would contain the number 3. There would be no DRIVE line for drive 2.

3. The first drive must always be used in a data base. If there is not enough room on the first drive to hold all of the tables need to generate the data base, an error message will be generated.

4. The DDL analyzer expects a file on the first drive having the name of the data base file specified in columns 8-19. If this is not the case, an error is generated.

5. For a multiple drive data base, the DDL analyzer requires only that the first drive have a file with the data base filename. Of course, the MDBS.DMS system requires that all referenced drives

have a file with the data base filename.

ITEM Line

The ITEM line defines a data item within the most recently defined record-type. Parameters define the item's type, size, replications and access levels.

Columns	Parameter Description
1-4	"ITEM" - line type
8-15	Item name - this field must contain a valid name for the item. The name must not start with a dollar sign ("$") or be blank.
17-20	Item type. Valid item types are: INT (Integer) - The item is stored under your host language format for integer variables. REAL - The item is stored under your host language format for real variables. BIN (Binary) - The item is an integer whose maximum value is limited by the item size specified. Binary variables may only be 1 or 2 bytes in length. LOG (Logical) - The item has a zero/one (true/false) value and is stored in one byte in the data base. CHAR (Character) - The item is a character string and is stored as a fixed length string, padded on the right with blanks if

necessary.

IDEC (Internal Decimal) - The item is stored under your host language format for internal decimal variables, (i.e., COBOL COMPUTATIONAL-3 fields).

XDEC (External Decimal) - The item is stored under your host language format for external decimal variables, (i.e., COBOL fields with a S999 picture).

21-24 Item size. This value is the number of bytes allocated in a record occurrence for the item. Size ranges vary according to the item type:

INTeger and REAL - The number of bytes specified should be the number of bytes used by your host language to store integer and real variables internally (see the MDBS.SYS manual). Using a size larger than this will result in wasted space in the data base; using a smaller size will result in a standard default (see the MDBS.SYS manual). Note that INT always will use the default value.

BINary - The size may be 1 byte or 2 bytes. A size of 1 allows values from -127 through 127 to be stored. A size of 2 allows values from -32767 through 32767 to be stored.

LOGical - The size may only be 1 byte.

CHARacter - The size should be equal to the largest string size to be stored in this item. No bytes need be allocated for string headers.

26-28 (optional) Read access level - a number between zero and 255 may be specified here to define the item's read access level. A item with a read access level of zero may be accessed by any valid data base user.

30-32 (optional) Write access level - a number between zero and 255 may be specified here to define the record's write access level. A record with a write access level of zero may be created, deleted or altered by any valid data base user.

36-43 (optional) Depending item name. If the last data item of a record type is a repeated item (see columns 46-47), the number of replications can be controlled by the value of another data item in this record. This other data item is called the depending (or depending-on) item. A depending item name can only be specified for the last data item of a record type. If no depending item name is specified and if the item is a repeated item, it is assumed to repeat a fixed number of times.

44-47 (optional) Replication factor. If the data item is a repeated item, the replication factor is specified in this field. If the item is a variable length item, the replication factor is the maximum number of times the item is allowed to repeat.

Notes:

1. If no read access level (columns 26-28) is specified, the read access level for the item will default to the read access level of the record containing the item.
2. The read access level for the item must not exceed the write access level.
3. The read access level for the item must be greater than or equal to the read access level for the record type containing the item.
4. The write access level for the item must be greater than or equal to the write access level for the record type containing the item.
5. A variable length data item can only be specified as the last item in a record type.
6. The read and write access levels of a variable length item type must be equal to the read and write access levels of its depending-on item type.

MEMBER Line

The MEMBER line is used to specify a member record type of a set. The format is:

Column	Parameter Description
1-6	"MEMBER" - line type
8-15	Record type name of a previously defined record type that is to be treated as a member record type.

Notes:

1. One or more MEMBER lines must follow the OWNER line(s) for each set in the DDL specification.

OWNER Line

The OWNER line is used to specify an owner record type of a set. The format is:

Columns	Parameter Description
1-5	"OWNER" - line type.
8-15	Record type name of a previously defined record type (or SYSTEM) that is to be treated as an owner record type.

Notes:

1. One or more OWNER lines must follow each SET line pair in the DDL specification.

2. SYSTEM owned sets may have only one owner.

PASSWORDS Line

The PASSWORDS line and section specifies the system's users, their passwords and access levels. The format for the PASSWORDS line is:

Columns	Parameter Description
1-9	"PASSWORDS" - line type

Following this line are the user definitions. The formats are:

Columns	
8-23	User identification.
26-28	User's read access level. The user can access (look-at) any record, item or set having read access level of this value or lower. The value must be in the range 0-255.
30-32	User's write access level. The user can write (modify) any item having a write access level of this value or lower. The value must be in the range 0-255.
36-47	User's Password.

Notes:

1. A user name need not be the actual name of an individual. For example, the user may be called "RECEIVING CLERK" or "SECRETARY."

2. The names and passwords are stored in the data base in an encrypted form.

RECORD Line

The RECORD line defines the start of a record description. Parameters include the record name and, optionally, read and write access levels for the record.

Column	Parameter Description
1-6	"RECORD" - line type
8-15	RECORD name - this field must contain a valid name for the record. The name must not start with a dollar sign ("$") or be blank.
26-28 (optional)	Read access level - a number between zero and 255 may be specified here to define the record's read access level. A record with a read access level of zero may be accessed by any valid data base user.
30-32 (optional)	Write access level - a number between zero and 255 may be specified here to define the record's write access level. A record with a write access level of zero may be created, deleted or altered by any valid data base user.

Notes:

1. The read access level for the record must not exceed the write access level.

2. A record-type named "SYSTEM" is predefined by the MDBS system; thus, no user-defined record can be named "SYSTEM"

SET Line

The SET line actually consists of two card. Set characteristics include the set type, set mode, usage factors, access levels and ordering information.

Columns	Parameter Description
1-3	"SET" - line type
8-15	Set name - this field must contain a valid name for the set. The name must not start with a dollar sign ("$") or be blank.
17-20	Set mode - Either "AUTO" or "MAN" may be specified. A set mode of AUTO indicates that whenever an occurrence of the member record type for this set is created, the record occurrence will be automatically added to the set. If MANual is specified, the user must explicitly add the record to this set.
22-24	Set type - One of the following set types must be specified: 1:N - Each owner record occurrence may own zero, one or more member record occurrences. Each member record may have at most one owner. (Note: This is a standard "one-to-many" set).

N:M - Each owner record occurrence may own zero, one or more member record occurrences. Each member record occurrence may be owned by more than one owner. (Note: This is a "many-to-many" set).

1:1 - Each owner record occurrence can own at most one member record occurrence. Each member record occurrence can be owned by no more than one owner record occurrence.

N:1 - Each owner record occurrence can own at most one member record occurrence. Each member record occurrence may be owned by one or more owner record occurrences.

26-28 (optional) Read access level - a number between zero and 255 may be specified here to define the set's read access level. A set with a read access level of zero may be accessed by any valid data base user.

30-32 (optional) Write access level - a number between zero and 255 may be specified here to define the set's write access level. A set with a write access level of zero may be modified (i. e., records may be added or removed) by any user.

34-36 Set usage factor. The set usage factor is a parameter which affects the trade off between processing time and disk space usage. A small

value (1 is the minimum value) optimizes disk utilization at the cost of increased overhead for each record insertion. A large value will result in less overhead for record insertions, but will result in a wastage of disk space. Sets which tend to have few occurrences (i.e., few owner record occurrences with one or more members) but are highly dynamic in terms of record insertions and removals should have a large value (such as 32) specified. Relatively static sets with many set occurrences should use a low value (such as 1 or 2). Choosing a set usage factor is a rather technical point and we recommend that you let it default. (The default value is currently 8). If too large a number is specified, a default value is computed giving the maximum number consistent with the data base page size.

8-13 (Line 2) Owner order - An order may be specified for the owner records of N:M or N:1 sets. Permissable set orderings are:

FIFO - When a new owner record occurrence is added to the set, it is logically placed as the last owner record occurrence in that set.

LIFO - When a new owner record occurrence is added to the set, it is logically placed as the first owner record occurrence in that set.

SORTED - When a new owner record occurrence is added to the set, it is placed in a sorted order in the set. The record occurrence with the smallest sort key value is logically first in the set. Sets can be accessed in descending order by use of the FLO and FPO commands.

IMMAT - The user is not concerned with the order of the record occurrences in the set. Use of the IMMATerial set order signals that MDBS.DMS may insert records into the set to maximize access efficiency.

17-24 (Line 2) Owner sort key - if the owner set order is SORTED, an item name may be specified as the sort key for the set. This must be a data item that has been defined in the owner record type. If this field is blank, the full record is used as a sort key.

30-35 (Line 2) Member order - An order may be specified for the member records of N:M or N:1 sets. Permissable set orderings are:

FIFO - When a new member record occurrence is added to the set, it is logically placed as the last member record occurrence in that set.

LIFO - When a new member record occurrence is added to the set, it is logically placed

as the first member record occurrence in that set.

SORTED - When a new member record occurrence is added to the set, it is placed in a sorted order in the set. The record occurrence with the smallest sort key value is logically first in the set. Sets can be accessed in descending order by use of the FLM and FPM commands.

IMMAT - The user is not concerned with the order of the record occurrences in the set. Use of the IMMATerial set order signals that MDBS.DMS may insert records into the set to maximize access efficiency.

PRIOR - When a new member record occurrence is added to the set, it is logically placed before the record indicated by the current member of the set. If the currency indicator for the current member of the set-type is null, the FIFO ordering is used.

NEXT - When a new member record occurrence is added to the set, it is logically placed after the record indicated by the current member of the set. If the currency indicator for the current member of the set-type is null, the LIFO ordering is

used.

39-46 (Line 2) Member sort key - if the member set order is SORTED, an item name may be specified as the sort key for the set. This must be a data item that has been specified in the member record type. if this field is blank, the full record is used as a sort key.

Notes:

1. If the set has a member sort key and has more than one member record type, then the set's sort key must exist as a data item type in each of the member record types. It must have the same name, type, and size in each of these record types.

2. If the set has an owner sort key and has more than one owner record type, then the set's sort key must exist as a data item type in each of the owner record types. It must have the same name, type, and size in each of these record types.


```
0010 FILES   DUM            2    512
0020 DRIVE   1       4
0030 DRIVE   2       2
0040 PASSWORDS
0050         PAM                          FROGGY
0060         GARY              255 255    646-46-1322
0070         ANYONE            004 004
0080 RECORD  A                 001 007
0090 ITEM    A1       CHAR 001 002 040
0100 ITEM    A2       LOG      001 007
0110 ITEM    A4       BIN  002
0120 ITEM    A5       REAL 012
0130 ITEM    A6       INT  006
0140 RECORD  B
0150 ITEM    B1       CHAR 003                          004
0160 ITEM    B2       LOG                                 2
0170 ITEM    B4       BIN  002                            4
0180 ITEM    B5       REAL 012                            3
0190 ITEM    B6       INT  007                            2
0200 RECORD  C
0210 ITEM    C0       INT  004
0220 ITEM    C1       BIN  002
0230 ITEM    C2       CHAR 001              C1        100
0240 RECORD  D                 009 012
0250 RECORD  E                 004 006
0260 ITEM    E1       CHAR 010 012 016
```

Sample DDL Input

FIGURE II.D.7 (Part 1)


```
0270 SET     S1        MAN   1:N 020 030 006
0280                                SORTED  A1
0290 OWNER   SYSTEM
0300 MEMBER  A
0310 SET     S2        MAN   1:N 005 032
0320                                SORTED  A1
0330 OWNER   B
0340 MEMBER  A
0350 SET     S3        MAN   N:M 007 007
0360         IMMAT                  FIFO
0370 OWNER   A
0380 MEMBER  B
0390 SET     S4        MAN   N:M
0400         SORTED    C1           LIFO
0410 OWNER   C
0420 MEMBER  C
0430 SET     S5        MAN   N:M 007 007
0440         NEXT                   PRIOR
0450 OWNER   A
0460 MEMBER  E
0470 SET     S6        MAN   1:1 007 007
0480
0490 OWNER   A
0500 MEMBER  B
0510 SET     S7        MAN   N:1 008 008
0520         FIFO
0530 OWNER   C
0540 OWNER   B
0550 MEMBER  B
0560 MEMBER  A
0570 MEMBER  E
0580 END
```

Sample DDL Input

FIGURE II.D.7 (Part 2)

6. Notes on Data Base Files

An MDBS data base is organized by a paging system and can be physically spread over up to 8 disk drives (floppy or hard disks may be used). The first drive of the user's system must have a data base file. The user can select other drives to participate in the data base.

During Data Definition analysis, a data base is initialized, and the first disk drive needs to be "on-line." However, whenever the data base is accessed by the Data Management System, all drives need to contain the appropriate disks.

Once the data base has placed information on a disk loaded on a particular drive, the disk must always be placed on that drive.

The maximum number of pages per drive is 8,191 and each page is logically restricted to 65,535 bytes. However, a page must also be able to fit in memory so a practical page size limit is probably under 4096 bytes.

In deciding on a page size, a user should take into account the trade-off between the number of pages that can be memory resident at one time and the number of times new pages will have to be read into memory. No simple rules can be given. However, the following ad hoc settings have been reliable in a number of applications:

1. Choose a page size so that at least 3 pages can be memory resident (see Section III.C.2).

2. A page size allowing 8 pages in memory usually is quite efficient.

3. In the absence of the above rules, choose a page size of 512 bytes.

In the appendix we present the results of an experiment which relates processing time to the number of memory resident pages. Since the number of memory resident pages is a function of both the page size and the amount of memory available to the DMS, the page size can play an important role in the execution efficiency of the data base system.

We conclude this section with a warning to BACK UP YOUR DATA BASE after any access that physically alters the data base.

E. Modes of Operation

1. Introduction

The purpose of MDBS.DDL is to describe a data base structure and, once the description has been completed, to initialize a data base. To accomplish this task MDBS.DDL provides features which allow the user to enter, alter and analyze his data base description. The data description analyzer attempts to initialize a data base using the data base description inputted by the user. This analysis is interrupted when a syntax or logical error is detected by the analyzer. At this point the user can quickly alter his description and again try to initialize the data base. This feature allows the user considerable flexibility and provides immediate feedback in building a data base description.

MDBS.DDL has three integrated modes of operation. They are:

1. Text Entry/Command Mode - where data descriptions are entered.

2. Line Editing Mode - where text lines are altered.

3. DDL Analyzer Mode - where a data base description is analyzed and a data base initialized.

In the next three sections each of these areas will be discussed in detail.

The normal procedure for using the MDBS.DDL package is to first enter a description of the data base using text entry and command features such as listing, deleting and saving; second, to edit the

text to correct errors; and third, to analyze the description and initialize a data base. If the analysis fails, the user automatically re-enters the Text Entry/Command mode so that changes can be made to the data base description.

Upon first entering MDBS.DDL, the user is in the Text Entry/Command mode. To enter the line editor or the DDL analyzer mode the user enters the appropriate command.

We will turn to a detailed discussion of the Text Entry/Command mode after the following general comments concerning conversation with MDBS.DDL.

Line-input editing conforms to the standard features of your operating system. While the system specific manual covers more on this, the following general comments are usually applicable. When entering a line (either of text or a command) the following keys are of special importance:

RETURN (ENTER) this key terminates a line of input

CONTROL-X this key interrupts the line entry and restarts the input.

CONTROL-P this key toggles the DDL output between the console and printer.

DELETE this key causes a physical backspace and character deletion in a line being entered.

CONTROL-H	same as DELETE.
BACKSPACE	same as DELETE.
CONTROL-C	this key interrupts a DDL operation and returns control to the entry mode.

When too many DELETE or BACKSPACEs are entered (i.e., the user has deleted all of his current line), these keys act as a line feed.

One key that operates the same in all three modes of operation is the ESCAPE key. Pressing this key always results in an exit from MDBS.DDL and returns control to the operating system.

The input buffer is limited to 80 positions. Hence a line longer than 80 characters cannot be entered. If the user attempts to type more than 80 characters, the message

*** LINE TOO LONG

will be displayed and the input line will be ignored. The user is still in Text Entry/Command mode.

An empty line (i.e., a line entered by pressing only the RETURN key) results in the message:

*** ERROR

As above, the user is still in Text Entry/Command mode.

2. Text Entry/Command Mode

In the Text Entry/Command mode, the user can enter both commands and text. Commands can be entered in lower or upper case. However, the user should be aware of the fact that file names are translated to upper case and that in the actual tables created for the data base, all names are stored as upper case. Specifically, user names and passwords, record names, item names and set names are stored in upper case. Of course, data to be stored in the data base via MDBS.DMS routines are maintained as specified by the user.

a. Text Entry

A line of text is preceded by a four (4) digit line number. Entered text is maintained in the order indicated by the line numbers. Examples of valid lines are:

```
0010 LINE OF TEXT

0020 MORE

7301 STILL MORE

0000 THIS IS OK

0732SO IS THIS

9999 HIGHEST LINE NUMBER
```

A line started with a line number consisting of fewer than 4 digits results in an error message:

```
***ERROR
```

and the line is ignored. The user is still in the Text Entry/Command mode.

If too much text is entered (i.e., if there is insufficient memory to hold all of the text) the following message is printed:

```
***OUT OF ROOM IN MEMORY
```

and the current line being entered is lost. However all text entered up to this point is still preserved. The user can expand the amount

of room available as outlined in the section "Important MDBS.DDL Addresses" (Section II.C.3).

If a line number has already been used, its subsequent re-use will remove the prior line.

b. Commands

In the following pages we describe each command. Commands are not preceded by blanks and must be followed immediately by a RETURN. If the command is not recognizable, the message:

***ERROR

is displayed. The user is still in Text Entry/Command mode when this happens.

(BYE, ESCAPE-key) Return to the Operating System

Purpose:

Exit from the program and return to the operating system.

Command Syntax:

BYE	Return to the operating system
ESCAPE-key	Return to the Operating System

Notes:

1. All text and variables are preserved so that the user can re-enter through the normal re-entry point.

2. The ESCAPE-key differs from the BYE command in that the BYE command is limited only to usage in the Text Entry/Command mode while the ESCAPE key can be used in all the modes of MDBS.DDL.

(D) Deleting Text

Purpose: Delete all or a portion of the entered text.

Command Syntax and Prompts:

D n	delete line n
D n,m	delete from line n to m (inclusive)
D n m	delete from line n to m (inclusive)
D	delete all the entered text
	results in computer prompt of:

DELETE ALL? (Y/N)

the text is deleted if a response of Y is given.

Examples of valid usage:

D 10	Deletes line 10
D 10, 50	Deletes lines 10,11,...49,50
D10 50	Deletes lines 10,11,...49,50
D 10 50	Deletes lines 10,11,...,49,50
D	Deletes all lines

DELETE ALL? (Y/N)

Y user responds with "Y"

Notes:

1. Commas or blanks serve as delimiters.
2. If there are no lines in the range to be deleted, the program returns to the Text Entry/Command mode.

3. After lines are deleted the program returns to normal Text Entry/Command mode.

(DDL) Data Definition Language Analyzer

Purpose:

This command is invoked to analyze the data base definition currently in text and, if no errors were encountered, to initialize a data base.

Command Syntax:

DDL invokes the data definition language analyzer.

Notes:

1. The DDL analyzer proceeds until either an error is encountered or a data base has been successfully initialized. Both types of terminations return the program to the Text Entry/Command mode. All text is preserved in memory.
2. The CONTROL-C key terminates the analysis process and returns the program to normal Text Entry/Command mode.
3. During the analysis, pressing any key (except CONTROL-C or ESCAPE) causes a pause until another key is pressed.
4. Before invoking the Analyzer, create the data base file on the primary drive. The file name should match the name given on the FILES line of the data base description being analyzed.
5. See Section II.E.4 for a more detailed discussion of the DDL Analyzer.

(DR, EN, FI, IT, ME, OW, PA, RE, SE) Data Description Format Commands

Purpose:

Display the format for the:

DRIVE

END

FILES

ITEM

MEMBER

OWNER

PASSWORDS

RECORD

SET

commands of a data base description. These are extremely useful while inputting text to describe a data base.

Command Syntax:

DR	For the drive section format
EN	For the end card format
FI	For the file section format
IT	For an item format
ME	For a set member format
OW	For a set owner format
PA	For the password section format
RE	For a record format
SE	For a set format

Examples of Valid Usage:

PA	Prints the password section format
PASS	Prints the password section format
PASSWORDS	Prints the password section format

Notes:

1. These commands do not affect the current text in any way. The user remains in Text Entry/Command mode.
2. Each of the commands can be entered by typing a portion of the section or card type name as long as at least the first two characters are typed. For example SE or SET results in the set card format but S alone will result in an error.
3. All the formats are displayed taking into account the the four digit line number and a leading blank (which is necessary in the data description text).

(E) Enter Line Edit Mode

Purpose:

This command is used to enter the line editor. The user is referred to the next section on LINE EDITING.

Command Syntax:

E	edit the entire text
En	edit the text starting at line n
En, m	edit the text starting at line n and ending at line m (inclusive)

Examples of Valid Usage:

E	edit all text starting at the the first line
E 100	edit text starting at line 0100
E100, 200	edit text between lines 0100 and 0200 inclusive
E 100, 200	edit text between lines 0100 and 0200 inclusive
E 100 200	edit text between lines 0100 and 0200 inclusive

Notes:

1. If there is no text within the range to be edited, the program returns to Text Entry/Command mode
2. If there is text within the range to be edited, the first such line is displayed. The user is now in the line editor mode.

(L) Listing Text

Purpose:

List the text entered by the user.

Command Syntax:

L	list all lines of text
L n	list line number n
L n,m	list text from line n to line m

Examples of Valid Usage:

L	list all lines of text
L10, 20	list lines 10 through 20
L10 20	list lines 10 through 20
L10 , 20	list lines 10 through 20
L10	list line 10
L0	list line 0

Notes:

1. Commas or blanks serve as delimiters.
2. During a listing, pressing any key (except CONTROL-C or ESCAPE) causes a pause until another key is pressed.
3. The CONTROL-C key terminates the listing process and returns the program to normal Text Entry/Command mode.

4. If there is no text to list the program remains in normal Text Entry/Command mode.
5. At the end of the listing the program returns to normal Text Entry/Command mode.

(N) Renumbering Text

Purpose:

Renumber all of the text entered by the user.

Command Syntax:

N renumbers all text giving the first line the number 0000 and increments by 0001.

Nn renumbers all text giving the first line the number n and increments by 0001.

Nn,m renumbers all text giving the first line the number n and increments by m.

Examples of Valid Usage:

N	First line will be 0000, second 0001, etc.
N 30	First line will be 0030, second 0031, etc.
N10,10	First line will be 0010, second 0020, etc.
N10, 10	First line will be 0010, second 0020, etc.
N0, 3	First line will be 0000, second 0003, etc.

Notes:

1. Commas or blanks serve as delimiters.
2. If an increment of zero is specified, the following message is printed:

   ***IMPROPER LISTING PARAMETERS

   and the text is renumbered using N 0001 0001.
3. If the parameters n and m are such that a line number greater than 9999 would result, the following message is printed:

***IMPROPER LISTING PARAMETER

and the text is numbered using:

L n 0001

At the completion of renumbering the program returns to normal Text Entry/Command mode.

(P) Print a Space Ruler

Purpose:

This command is used to produce a space ruler to act as a guide for text entry. For example, on a 64 character output device this command results in

```
  1...5...10...15...20...25...30...35...40...45...50...55
```

so that text can be entered as:

```
  1...5...10...15...20...25...30...35...40...45...50...55
0020 TEXT
```

Command Syntax:

P	Produces a space ruler.

Notes:

1. This command does not affect the current text in any way. The user remains in Text Entry/Command mode.

(R) Reading a Text File

Purpose:

Allows the user to read a previously saved text file.

Command Syntax and Prompts:

R	read a previously saved file
FILENAME	prompt from computer
name	the user should respond with a valid file name (fully qualified)
	on a successful read the computer responds with:
xxxxx BYTES	

Example of Valid Usage:

R

FILENAME

name

463 BYTES

Notes:

1. Any text in memory is replaced by the file brought in by a successful read.
2. If a file is not successfully read the message:

   ***ERROR

   is displayed (in addition to any messages the operating system may print). Some read errors may result in a return to the operating system in which case the user may re-enter MDBS.DDL

through the standard re-entry point.

3. If a file is not successfully read, text present in memory may still be intact, depending of course on the nature of the read error.

4. Normal causes for read errors include the absence of the indicated file on the indicated disk drive. Other standard problems causing read errors may also be encountered.

5. To escape from a FILENAME prompt the user may enter a null line (a simple RETURN). This will result in the message

   ***ERROR

   but will return the user to the normal Text Entry/Command mode.

6. The response to a successful read:

   xxxxx BYTES

   gives the total number of bytes transferred.

6. If there is not enough memory available for the text to be read-in the following message is displayed:

   ***OUT OF ROOM IN MEMORY

   and reading discontinues. Text in memory before the R command remains in memory and the program returns to Text Entry/ Command mode.

(W) Writing a Text File

Purpose:

Allows the user to write a text file.

Command Syntax and Prompts:

W	write the current text onto a file
FILENAME	prompt from computer
name	the user should respond with a valid file name (fully qualified)
	on a successful write the computer responds with:
xxxxx BYTES	

Example of Valid Usage:

W

FILENAME

name

463 BYTES

Notes:

1. A successful write will produce an image of the text on the appropriate file. The text still resides in memory. At the completion of the WRITE operation the program returns to normal Text Entry/Command mode.
2. If a file is not successfully written the message:

   ***ERROR

   is displayed (in addition to any messages the operating system

may print). Some write errors may result in a return to the operating system in which case the user may re-enter MDBS.DDL through the standard re-entry point.

3. Normal causes for write errors include the absence of the indicated file on the indicated disk drive. Other standard problems causing write errors may also be encountered.
4. To escape from a FILENAME prompt the user may enter a null line (a simple RETURN). This will result in the message

   ***ERROR

   but will return the user to the normal Text Entry/Command mode.
5. The response to a successful write

   xxxxx BYTES

   gives the total number of bytes transferred.

3. Line Editing

In the line editing mode, current text can be altered within a line without retyping the line. The line editing mode is entered through the E command of the Text Entry/Command mode. Once in the line editor, the current line being edited is displayed. At this point, the user may advance to the next line of text or may make changes in the current line. A given line can be repeatedly altered before advancing to the next line.

As in the Text Entry/Command mode, all communication with the system can be performed in lower or upper case and all special keys retain their normal functions.

We now turn to a detailed discussion of the line editing features and commands. Note that the entry of an improper command syntax results in a prompt of:

?

When this occurs the user is still in line editor mode and can give a correct response.

(RETURN key) Move Through Text

Purpose:

This allows the user to advance from the current line of text to the next line of text and to make the next line of text current.

Command Syntax:

Press RETURN key to move to the next line of text

Notes:

1. When entering the line editor mode through the E command of the Text Entry/Command mode, the user specifies a target region for editing. Once the end of the region is encountered and the user presses the RETURN key, the program returns to Text Entry/Command mode.
2. After pressing the RETURN key, the new current line of text is displayed.

(C) Change a Line of Text

Purpose:

This command is used to alter a string of text within a line.

Command Syntax:

CdSOURCEdNEWd — where "d" is any delimiter, SOURCE is the original string and NEW is the replacement string.

Example of Valid Usage:

Assume the following is being edited:

```
0010 HTIS IS TWO DEMANSTRATE THE LINE LINE EDTOR
```

C/HT/TH/ results in:

```
0010 THIS IS TWO DEMANSTRATE THE LINE LINE EDTOR
```

C.TWO.TO results in:

```
0010 THIS IS TO DEMANSTRATE THE LINE LINE EDTOR
```

CQMAQMO results in:

```
0010 THIS IS TO DEMONSTRATE THE LINE LINE EDTOR
```

C. LINE. results in:

```
0010 THIS IS TO DEMONSTRATE THE LINE EDTOR
```

C/ED/EDI results in:

```
0010 THIS IS TO DEMONSTRATE THE LINE EDITOR
```

Typing:

```
S
```

returns the program to Text Entry/Command mode.

Notes:

1. The first character following the C is used as the delimiter.
2. The last delimiter is optional. The RETURN key also acts as the last delimiter.
3. If the source string is not found, the computer responds with:

   ?

   and a proper response can then be given.
4. The line number cannot be altered by the line editor.

(S) Leaving the Editor Mode

Purpose:

To return the program to Text Entry/Command mode.

Command Syntax:

S	return to Text Entry/Command mode

(C) Repeated Changes

Purpose:

To make the same changes within and across lines.

Method:

When a change is entered, it is "remembered". The next change replaces the remembered change and becomes the new "remembered" change. Upon entering the text editor, the "remembered" change is the default of

C///

Entering a C followed by a RETURN invokes the last remembered change. This can be done within a line or, after advancing lines, within a new line.

Command Syntax:

C make remembered change

Examples of Valid Usage:

Suppose the following text is to be edited:

```
0010 PRODUCT 64KGB IS NOT NEW
0020 64KGB AND 64KGB DERIVATIVES
0030 HAVE BEEN AROUND FOREVER.
0040 ... 64KGB REPORT.
```

and the user entered E in the Text Entry/Command mode. This results in:


```
     0010 PRODUCT 64KGB IS NOT NEW.
```

being displayed. At this point the remembered change is:

```
C///
```

Suppose 64KGB is to be replaced by T47S. The user may enter:

```
C.64KGB.T47S
```

which will result in:

```
     0010 PRODUCT T47S IS NOT NEW.
```

and the current remembered change is:

```
C.64KGB.T47S
```

Pressing the RETURN key will result in:

```
     0020 64KGB AND 64KGB DERIVATIVES
```

Pressing C and then RETURN results in:

```
     0020 T47S AND 64KGB DERIVATIVES
```

since C.64KGB.T47S was remembered. Repeating with C and RETURN gives:

```
     0020 T47S AND T47S DERIVATIVES
```

Pressing RETURN gives:

```
     0030 HAVE BEEN AROUND FOREVER
```

Pressing RETURN gives:

```
     0040 ... 64KGB REPORT.
```

Entering C and RETURN gives:

```
     0040 ... T475 REPORT.
```

Notes:

1. A remembered change will not be forgotten until either another properly entered change is submitted or the program returns to Text Entry/Command mode.

4. DDL Analyzer

In the DDL analyzer mode, text containing a data base description is first analyzed and then, if error free, used to initialize a data base.

While a data base may be spread over several disks, (as explained in the FILES section of the Data Description Language discussion), MDBS.DDL expects to initialize a file on the first physical drive. Only this drive needs to have the indicated file present during the DDL analysis. Of course, during MDBS.DMS operations, all files need to be active.

If a data description is successfully analyzed and a data base initialized, the message

DDL PROCESSING COMPLETED

is displayed. The program then returns to Text Entry/Command mode. All text has been preserved.

If an error is detected during the analysis, an error message is displayed and the program returns to the Text Entry/Command mode. All text is preserved.

During a DDL analysis, pressing any key (other than CONTROL-C or ESCAPE) results in a pause until a key is again pressed. This is useful for purposes of controlling the output displayed.

CONTROL-C interrupts the analysis and returns the program to the Text Entry/Command mode. The ESCAPE key returns control to the operating system.

The DDL analyzer builds tables in memory beyond the text region. If there is insufficient room in memory to complete all of the tables, the message

***OUT OF ROOM IN MEMORY

is displayed. As with all errors, control is returned to the Text Entry/Command mode and all text is preserved. The user should refer to Section II.C.3 for information on expanding available memory.

While analyzing the data description, each line of description is printed (without line numbers).

In the following, a list of errors detected by the DDL analyzer is given along with explanations and possible causes.

*** A DEPENDING ON ITEM CANNOT BE A REPEATED FIELD

Line Type: ITEM

Columns: 36-43

Explanation:

A "depending on" item was specified in columns 36-43 for the item currently being processed. The depending on item was defined as a repeated field, which is not permitted.

Possible Causes:

1. Typographical error.
2. Incorrect item specified as depending item.
3. Data base design error.

*** A VARIABLE LENGTH ITEM MUST BE LAST IN A RECORD

Line Type: ITEM

Explanation:

A variable length item (i.e., one with an entry in columns 36-43) must be the last item in a record. The ITEM line being processed immediately follows a variable length item. Only one variable length item may appear in a record, and it must be the last item.

Possible Causes:

1. ITEM lines out of order.
2. Missing RECORD line.
3. Data base design error.

*** CANNOT WRITE TO DISK

Explanation:

A request to the operating system was made by the DDL analyzer to write to a file. The operating system returned an error flag.

Possible Errors:

1. Refer to the standard causes for disk errors.
2. A file with the data base name cannot be found on drive 1.

*** CAN'T HAVE OTHER OWNERS WITH SYSTEM

Line Type: OWNER

Columns: 8-13

Explanation:

Two or more owner lines were specified for this set and one of the lines defined SYSTEM as an owner. If SYSTEM is the owner record for a set, then no other owner record types may be defined for that set.

Possible causes:

1. Specified OWNER instead of MEMBER on a line.
2. Data base design error.

*** DEPENDING ON ITEM MUST BE A TWO BYTE BINARY VARIABLE

Line Type: ITEM

Columns: 36-43

Explanation:

A "depending on" item was specified in columns 36-43 for the item currently being processed. The depending on item is a binary variable, but does not have length two as required.

Possible Causes:

1. Typographical error.

2. Incorrect item specified as depending item.

3. Data base design error.

*** DEPENDING ON ITEM MUST BE BINARY

Line Type: ITEM

Columns: 36-43

Explanation

A "depending on" item was specified in columns 36-43 for the item currently being processed. The depending on item must be BINary with a length of 2 bytes.

Possible Causes:

1. Typographical error.

2. Incorrect item specified as depending item.

3. Data base design error.

*** DEPENDING ON ITEM NOT PREVIOUSLY DEFINED IN THIS RECORD

Line Type: ITEM

Columns: 36-43

Explanation:

The item specified in columns 36-43 of the current ITEM line was not defined on a prior ITEM line for the current record type.

Possible Causes:

1. The ITEM line for the depending on item is missing or out of sequence.
2. The name of the depending item was misspelled when the item was defined.
3. The item name is columns 36-43 is misspelled.
4. The item name for the depending-on item does not start in column 36.
5. Typographical error.
6. Data base design error.

*** DUPLICATE ITEM NAME IN RECORD

Line Type: ITEM

Columns: 8-15

Explanation:

The same name has been given to two ITEMs in the same record.

Possible Causes:

1. A redundant line of text.
2. Typographical error.
3. Missing RECORD line before the replicated ITEM.
4. The second ITEM line should have been a RECORD or SET line.
5. Data base design error.

Notes:

1. An ITEM may be replicated across RECORDs. That is, the same ITEM name may appear in different RECORDs.

*** DUPLICATE RECORD NAME

Line Type: RECORD

Columns: 8-15

Explanation:

Two RECORD types have the same name.

Possible Causes:

1. A redundant line of text.
2. Typographical error.
3. The second RECORD line should have been an ITEM or SET line.
4. Data base design error.

© COPYRIGHT 1979, 1980, Micro Data Base Systems, Inc.

*** EXPECTING A NUMBER IN A FIELD

Line Type: See Below

Columns: See Below

Explanation:

A non-numeric character was entered in a numeric field. Valid numeric characters are the digits 0-9 and blanks.

Possible Causes:

1. Typographical error.

2. Number entered in wrong columns.

Note:

Numeric fields are found in the following columns:

PASSWORDS line	FILES line
26-28	22-22
30-32	26-29
DRIVE line	**RECORD line**
8-8	26-28
	30-32
ITEM line	**SET line**
21-24	26-28
26-28	30-32
30-32	34-36
44-47	

*** EXPECTING A RECORD, ITEM, OR SET LINE

Explanation:

Something other than a RECORD, ITEM, or SET line was encountered after the PASSWORDS or FILES section or within the RECORD section.

Possible Causes:

1. Sections out of order.
2. Typographical error.
3. Missing RECORD, ITEM or SET line.
4. Columns 1-6 of the line were blank.

*** EXPECTING AUTO OR MAN

Line Type: SET

Columns: 17-20

Explanation:

Sets may be either AUTO or MANual. One of these two set modes was not specified.

Possible Causes:

1. Typographical error.
2. The word AUTO or MAN did not start in column 17.
3. No set mode was specified.

*** EXPECTING SET OR END LINE

Explanation:

Something other than a SET or END line was encountered while in the SET section.

Possible Causes:

1. Sections out of order.
2. Typographical error.
3. Missing SET or END line.
4. Columns 1-6 of line were blank.

*** FILE HAS NOT BEEN CREATED: PLEASE DO SO

Explanation:

The DDL analyzer has attempted to initialize a data base, but the data base file does not exist on the disk on the first physical drive.

Possible Errors:

1. The user has not yet created a file with the appropriate name.
2. The file exists on a disk located on some drive other than the first drive.
3. The filename may have been misspelled in the FILES section or in the DDL prompt if there is no FILES section.

*** IMPROPER DRIVE NUMBER

Line Type: DRIVE

Column: 8

Explanation:

A drive number was either zero or greater than the number of drives allowed. The number of drives allowed was specified on the FILES line.

Possible Errors:

1. The FILES line has too small an allocation.
2. Column 8 of the current DRIVE line was left blank.
3. Typographical error.

*** INCORRECT MEMBER ORDER

Line Type: SET (line #2)

Columns: 30-35

Explanation:

The ordering specification for the members of the current set is not valid. Defined orders are:

FIFO	first in-first out
LIFO	last in-first out
NEXT	insert "after" current member
PRIOR	insert "prior to" current member
IMMAT	ordering is immaterial
SORTED	set is sorted

Possible Causes:

1. Typographical error.

2. The ordering information does not start in column 30.

Notes:

1. If columns 30-35 are blank, IMMAT is assumed.

2. It is not meaningful to specify an member ordering for N:1 or 1:1 sets. However, if an order is given, it will be checked for validity.

*** INCORRECT OWNER ORDER

Line Type: SET (line #2)

Columns: 8-13

Explanation:

The ordering specification for the owners of the current set is not valid. Defined orders are:

FIFO	first in-first out
LIFO	last in-first out
NEXT	insert "after" current owner
PRIOR	insert "prior to" current owner
IMMAT	ordering is immaterial
SORTED	set is sorted

Possible Causes:

1. Typographical error.
2. The ordering information does not start in column 8.

Notes:

1. If columns 8-13 are blank, IMMAT is assumed.
2. It is not meaningful to specify an owner ordering in 1:N or 1:1 sets. However, if an order is given, it will be checked for validity.

*** INVALID ITEM TYPE

Line Type: ITEM

Columns: 17-20

Explanation:

The type specified for the item being processed is not one of the types listed below:

BIN	Binary
CHAR	Character
INT	Integer
LOG	Logical
REAL	Real

Possible Causes:

1. Typographical error.
2. The item type name does not start in column 17.
3. No item type was specified.

*** INVALID SET CHARACTERISTICS

Line Type: SET

Explanation:

An inconsistency was detected by the DDL Analyzer while generating the set description tables for the set being processed. This error is internal to the DDL Analyzer code and should not normally occur. If it can be determined that the possible causes listed below are not responsible for this error, please contact MDBS for assistance.

Possible Causes:

1. Improper user patch. Check MDBS manual for proper patch procedures.

2. A hardware or software malfunction has caused the set descriptors to become inconsistent.

*** INVALID SET TYPE

Line Type: SET

Columns: 22-24

Explanation

One of the four possible set types has not been specified on the SET line. Valid set types include:

1:N	One owner record occurrence may be associated with zero or more member record occurrences (Standard CODASYL set)
N:M	Each owner record occurrence may have more than one member record occurrence and each member record occurrence may have more than one owner record occurrence (many-to-many).
1:1	Each owner record occurrence may be associated with at most one member record occurrence.
N:1	Each member record occurrence may be associated with one or more owner record occurrences; each owner may have at most one member record occurrence.

Possible Causes:

1. Typographical error.

2. The set type specification does not start in column 22.

3. No set type was specified.

*** ITEM READ OR WRITE ACCESS LESS THAN RECORD'S

Line Type: ITEM

Columns: 26-28 or 30-32

Explanation:

Either the read or write access level for the item being processed is less than the corresponding access level for the record containing this item. This usually indicates that an error has been made in the record or item access level specifications.

Possible Causes:

1. Typographical error.

2. Data base design error.

Note:

If the access levels for an item are left blank, they will default to the record access levels.

*** KEY UNDECLARED

Line Type: SET (line #2)

Columns: 17-24 or 39-46

Explanation:

A sort key was specified which does not appear as an item type in the owner (for sorted owner type) or member (for sorted member order) record type.

Possible Causes:

1. Typographical error.
2. Sort key name does not start in proper column.
3. Data base design error.

Notes:

1. It is not meaningful to specify a sort key unless the set order is SORTED. If the set order is not SORTED and a sort key is specified, the sort key name will be checked for validity, but will not be used.
2. It is permissable to not specify a sort key for a sorted set. If this is done, the full record is used as the sort key.

*** MAX LENGTH FOR BINARY VARIABLE IS 2

Line Type: ITEM

Columns: 22-24

Explanation:

An item of type binary (BIN in column 17-20) has been encountered with an item size (columns 22-24) larger than 2. Binary variables may be only one or two bytes in length.

Possible Causes:

1. Typographical error.
2. Data base design error.
3. Item size not right justified in columns 22-24.

*** MAXIMUM RECORD SIZE TOO LARGE TO FIT ON PAGE

Explanation:

This error is displayed after the END line has been reached when, in the process of initializing the data base, the DDL Analyzer discovers that a record is too large to fit on a data base page. The data base page size is defined on the FILES line. Each page also has a 10 byte page header.[1]

Possible Causes:

1. Page size on FILES line (columns 26-29) too small.

2. An item in a record has an item length and specification count too large.

[1] It is possible that future releases of MDBS.DMS will require slightly larger record and page headers. We recommend that you make allowances for this when you select the page size and record sizes for your data base.

*** NAMES CANNOT START WITH A $ OR BE BLANK

Line Type: RECORD, ITEM, SET

Columns: 8-15

Explanation:

All records, items and sets must be given a non-blank name. Names starting with a dollar sign are reserved for use by MDBS processors.

Possible Causes:

1. Typographical error.
2. Incomplete specification.
3. Data base design error.

*** NO MEMBER AND/OR OWNER LINE FOR A SET

Explanation:

A set declaration was encountered which did not contain at least one owner line and at least one member line.

Possible Causes:

1. Typographical error in word OWNER or MEMBER.
2. Omitted OWNER or MEMBER line.
3. MEMBER line specified before OWNER line.

*** NO PAGES ALLOCATED TO A FILE

Line Type: DRIVE

Columns: 12-15

Explanation:

A DRIVE line specified a file length (on that drive) of zero pages.

Possible Causes:

1. Conceptual error (see note below).
2. Missing DRIVE line.
3. Typographical error.

Notes:

1. A file of length zero can never be used by the data base management system. Hence this error points out a null request. If the user has entered such a line in the mistaken belief that the data base must be allocated on drives in a contiguous fashion, the following illustration may be of use:

```
DRIVE  3   20
DRIVE  1   20
```

This example states that drives 1 and 3 will hold the data

base (notice drive 2 will not be used) and that each will hold, at maximum, 20 pages.

*** NOT ENOUGH ROOM ON DRIVE 1

Explanation:

The DDL analyzer has attempted to initialize a data base, but the amount of information needed to be placed on the file on drive 1 exceeds the amount of room available on this file.

Possible Errors:

1. A specification line for DRIVE 1 (in the FILES section) does not provide for enough pages on the drive. Provide for more room if possible.

*** NUMBER LARGER THAN 255

Line Type: See below

Explanation:

A number larger than 255 has been encountered. The following numeric fields are limited to a maximum value of 255:

PASSWORDS	RECORD
26-28	26-28
30-32	30-32

ITEM	SET
26-28	26-28
30-32	30-32
	34-36

*** PAGE LENGTH MUST BE DIVISIBLE BY 256

Line Type: FILES

Columns: 26-29

Explanation:

The data management system assumes that page sizes are multiples of 256. Hence a page length must be evenly divisible by 256.

Possible Errors:

1. A page length of zero generates this message. Hence columns 26-29 may be blank.

2. Typographical error.

3. Improper specification.

*** PASSWORD LINE EXPECTED

Explanation:

A PASSWORDS line must be the first line of the data description.

Possible Causes:

1. Sections out of order.
2. Typographical error.
3. PASSWORDS section missing or PASSWORDS lines missing.
4. Columns 1-6 of line were blank.

*** PASSWORD ENTRY OR RECORD LINE EXPECTED

Explanation:

A line following the PASSWORDS line has been encountered which is not a password entry or a RECORD line.

Possible Causes:

1. "RECORD" misspelled on RECORD line.
2. Columns 1-7 of password entry line non-blank.
3. Sections out of order.

*** PREMATURE END OF INPUT

Explanation:

The data description is incomplete.

Possible Causes:

1. The END line was not present.

2. Not all sections of a data base description are present, or, if present, possibly are out of order. The sections (in appropriate order) are:

```
PASSWORDS
FILES (optional)
                    DRIVE (optional)
RECORD
                    ITEMS (optional)
SET
                    OWNER
                    MEMBER
END
```


*** READ ACCESS GREATER THAN WRITE ACCESS

Line Type: PASSWORDS, RECORD, ITEM, SET

Columns: 26-28, 30-32

Explanation:

The read access level specified (in columns 26-28) was greater than the write access level specified (columns 30-32). Since larger access levels indicate more restrictive access, the read access is more highly restricted than the write access, which is not meaningful.

Possible Causes:

1. Typographical error.

2. Data base design error.

3. Omission of an access level entry.

*** RECORD ACCESS GREATER THAN SET'S

Line Type: OWNER, MEMBER

Explanation:

The record specified on the OWNER (MEMBER) line being processed has a higher (i.e., more restrictive) read or write access level than the set to which it belongs.

Possible Causes:

1. Typographical error.
2. Data base design error.
3. Omitted access level.

*** RECORD NOT FOUND

Line Types: OWNER, MEMBER

Columns: 8-13

Explanation:

The record type specified on columns 8-13 was not previously defined.

Possible Causes:

1. Typographical error.
2. Record name does not start in column 8.
3. No record name specified.
4. Record name mis-specified on RECORD line.
5. Data base design error.

*** SECOND LINE OF SET DEFINITION INVALID

Line Type: SET

Columns: 1-6

Explanation:

The set description actually consists of two line-images. The first has the word SET in columns 1-6 and the second has blanks in these columns. A valid first line of the set description was processed, and a line which was non-blank in columns 1-6 has been encountered after it.

Possible Causes:

1. Omission of second line of set description.

2. Typographical error.

*** SORT KEY NOT IN RECORD

Line Type: OWNER, MEMBER

Columns: 8-13

Explanation:

A sort key was specified for the owner (member) of this set, but the record on this OWNER (MEMBER) line does not contain the sort key.

Possible Causes:

1. Typographical error.

2. Data base design error.

*** SYSTEM CAN'T BE A MEMBER

Line Type: MEMBER

8-13

Explanation:

SYSTEM was defined as a member of a set. SYSTEM is only allowed to be the owner of a set.

Possible Causes:

1. Data base design error.

III. MDBS.DMS

A. Introduction

In this section, we present Micro Data Base Systems' Data Management System which we call MDBS.DMS (for Micro Data Base Systems' Data Management System). A user controls the Data Management System through the Data Manipulation Language (DML).

In part B we list several features of the system which are described in more detail elsewhere. In section C the user is instructed on how to "bring-up" MDBS.DMS and how to modify the package.

In the MDBS system, various data manipulation language commands are used. In section D we discuss each (DML) command in detail. Section D also provides examples of the use of these commands. It is strongly recommended that these examples be read before the detailed information of Section III.D.3 is studied.

Understanding of the DML commands can be expedited by first examining the following commands. These commands enable a user to actually use a data base, and serve as a basis for gaining an overview of the system:

1.	DEFINE	6.	FNM
2.	OPEN	7.	GFM
3.	CRS	8.	SOM
4.	AMS	9.	SMM
5.	FFM	10.	CLOSE

B. Features

MDBS.DMS allows the user to store and retrieve data in the data base and to establish and use the various set relationships defined in the MDBS Data Definition Language. All details of physically managing the data are handled by the MDBS.DMS system -- specifically, the tasks of reading, writing and managing disk storage are handled by MDBS.DMS.

MDBS.DMS supports the following features for data base management work:

1. Convenient host language calling sequence.
2. Powerful data block features for communication with host language variables- a must for non record oriented languages.
3. Records may be fixed or variable length.
4. Data items may be character, integer, real (floating point), logical, or binary.
5. Sets may be one-to-many, many-to-many, one-to-one, or many-to-one, instead of merely one-to-many.
6. Sets may be ordered as SORTED, FIFO, LIFO, NEXT, PRIOR or IMMATerial. The immaterial ordering allows the system to achieve certain economies in accessing the data base.
7. Automatic or manual insertion of records into sets is supported.
8. Read and write access protection via passwords at the item, record and set levels of organization.

C. Getting Started with MDBS.DMS

1. Relocating MDBS.DMS

For interpreted languages it is often desirable to be able to relocate the data management system to a position in memory other than that provided by Micro Data Base Systems. In particular, it is frequently useful to mesh MDBS.DMS with the host language. For example, if BASIC is the host language, the user may wish to physically append MDBS.DMS to his BASIC interpreter and store the appended version as, say, BASDMS. Undoubtedly, MDBS.DDL will have to be relocated for this purpose.

For this purpose we have provided a relocatable form of the data management system and a relocator so that an executable form of the system can be ORGed to any place in memory. Refer to the system specific manual for further information.

2. Personalizing And Patching MDBS.DMS

MDBS.DMS consists of a program region, table region, page region and defined block region (in non record oriented languages). The last three are contiguous and dynamically allocated.

To roughly compute the size of the table region, add the following partial sums:

```
   42
 + 25 * Number of Record Types
 + 21 * Number of Items
 + 42 * Number of Sets
 +  3 * Number of Owner Lines
 +  3 * Number of Member Lines
 + 30 * Number of Password Lines
 +  2 * Total Number of Pages Allocated
 + 44 * Number of Disk Drives
```

We will refer to this value as T.

The page region consists of at least one block of memory equal in size to the page length specified in the FILES section of the data base description. This area is allocated depending on a number of factors listed below. An extremely important factor is the last word of memory specification, which is discussed later in this section.

Finally, in non record oriented languages the host language user defines data-blocks for transferring information into (out of) the data base from (to) his program variables. The size of the block region can be computed by:

```
   12 * Number of defined blocks
 +  2 * Total number of variables occurring in the defined
        blocks.
```


We will refer to this value as B. For record oriented languages (COBOL, PASCAL, PL/I, etc.), use B=0.

There are several addresses that the user should be aware of in MDBS.DMS. Two of these give the first and last word of memory that the data management system can use. Call these values FW and LW. Given FW and LW and the size of the user tables, the number of pages available for buffering data to and from memory is computed by MDBS.DMS as follows:

$$\frac{(LW - FW - T - B)}{(Page\ Size + 3)} = \#\ of\ Pages$$

Whenever a new data-block is defined in a user program, the number of pages is recomputed by the data management system. If this number is less than one, an error is returned to the user.

In the Appendix we present the results of an experiment which relates processing time to the number of memory resident pages. Since the number of memory resident pages is a function of both the page size and the amount of memory available to the DMS, the number of pages available can play an important role in the execution efficiency of the data base system.

Below we discuss addresses of interest to the user of MDBS.DMS. The user may alter these. A brief description of each follows:

(a) Data Management System Entry Point(s)

The user enters at these points to execute all of the DMS commands.

(b) Last Word of Memory

The address stored here gives the last available word of memory that MDBS.DMS may use.

(c) First Word of Memory

The address stored here gives the first available word of memory that MDBS.DMS may use.

(d) Operating System Entry Points

Different operating systems handle disk read and writes in a variety of ways. To keep the manual free of system specifics, such information relevant to your system is published in the system specific manual. Patches to non-standard routines should be made in accordance with the instructions found there.

D. MDBS Data Management System

1. Introduction and Definitions

The Micro Data Base Systems' Data Management System supports a Data Management Language (DML) which is comprised of a large set of subroutines. When called from a host program, the DML routines request the DMS to perform certain operations with the data base. These operations include finding, adding and deleting record occurrences, fetching and putting item occurrences, and setting currency indicators. See Appendix 2 for a brief description of the DML functions.

In this discussion, a data item in a record type will be referred to as a "field"; the fields are ordered, and their values occupy consecutive words in memory in a record occurrence. Consider the record type:

```
RECORD EMPLOYEE
ITEM   NAME      CHAR  20
ITEM   NUMBER    INT    8
ITEM   WAGE      REAL   8
ITEM   TAX       REAL   8
```

In an occurrence of this record type, a 20 byte value for NAME is first, followed by an 8 byte value for NUMBER, etc.

A DML command is actually a subroutine call. The FMSK command (for example) has the BASIC format:

```
E0 = CALL (A0, "FMSK, set-type, data-block-name")
```

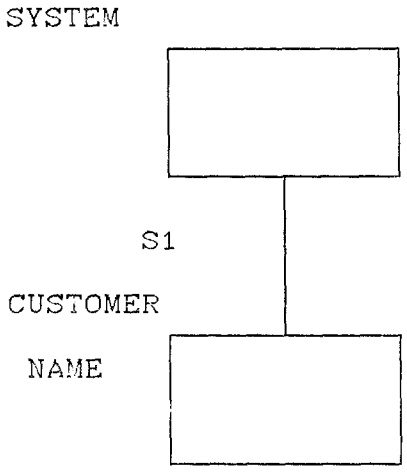
Most commands (the DEFINE and EXTEND commands are the only exceptions) pass a string which indicates the command to be processed and the

parameters for the command. The commands return a value (here to variable E0) which is set to zero if the command executes properly and non-zero upon detection of any error or abnormal status. A list of possible errors is given later in this manual. The A0 refers to the address of the entry point of the DMS.

o A DATA-BASE KEY (refered to as the "key") is an address associated with a record occurrence indicating where in the data base that record occurrence is physically located. Each record occurrence has a unique key associated with it. No two record occurrences can have the same key.

o A CURRENCY INDICATOR is a special key maintained by the DMS containing the data base address of a particular record occurrence. There are two currency indicators for each set-type, one indicating the current owner record occurrence and the other indicating the current member record occurrence. So, for each set, there is a "current owner" and a "current member." There is also a currency indicator for each record-type, specifying the current record occurrence of that record-type. A third type of currency indicator maintained by the DMS is the current of run unit. The current of run unit is simply the data base key of the record last referenced by the DMS. Certain DML commands use the currency indicators to specify on which record occurrence to operate, and others will change appropriate currency indicators; the currency indicators will be changed to show that another record occurrence is now current.

For example, if SET S1 is sorted on NAME and has owner SYSTEM and member CUSTOMER,



then the Find Member based on Sort Key command:

```
E0 = CALL (A0, "FMSK, S1, N0")
```

causes the following to happen: The system uses a binary search on the member record occurrences of set S1 until it finds one with the NAME field equal to the value of variable N0. It then saves the data-base key of that record in the currency indicator for the member record of set S1. That is, the system makes that record the current member of S1.

The statement:

```
E0 = CALL (A0, "DRM, S1")
```

(the Delete Record based on current Member command) instructs the system to delete the current member record occurrence of set S1. The system gets the key from the currency indicator for the member record of set S1 and deletes the record beginning at the location indicated by the key. The sequence of the above two commands would cause the record with a sort key value equal to variable N0 to be deleted.

(Actually, in the above discussion NO is the name of a data block pointing to a user variable called NO.)

Every time the data base system is used, an initialization process must take place at the beginning of execution. This is handled via the OPEN command. With the possible exception of the DEFINE and EXTEND commands, the OPEN command must be the first DML command executed each time MDBS.DMS system is used. The format of the command is:

```
E0 = CALL (A0, "OPEN, OPENLIST")
```

OPENLIST is a data block pointing to four user variables. In this case the four variables must be character strings containing:

1. The data base name, for example:

```
F$ = "CUSTOMER"
```

2. The user's name, for example:

```
N$ = "GEORGE SHELL"
```

3. The user's password, for example:

```
P$ = "IGLOO"
```

4. The data base access status, for example:

```
S$ = "MODIFY"
```

   This allows the user to modify (i.e., write onto) the data base. Any other string would restrict the user to read-only mode.

At the beginning of execution of MDBS.DMS, all currency indicators are initialized. Execution of the OPEN command has the following effect: All sets which have been defined in the DDL as having SYSTEM as their owner record-type will have their currency indicator for

current owner equivalenced to SYSTEM. These sets will initially be the only ones with current owners, and almost all DML commands work with a currency indicator to obtain a particular record. So it can be seen that the OPEN command and a set with owner SYSTEM is required to get "a foot in the door" to begin fetching or storing record occurrences.

Just as OPEN must be executed as the first DML command, CLOSE must be executed as the last DML command. CLOSE has no parameters.

The owner-member relationships of sets is a very important concept to understand. In the examples that follow, diagrams will be used similar to those in the DDL discussion. Figure III.D.1 is a representation of set S1 having owner SYSTEM and member A and set S2 having owner A and member B. (In all future diagrams, the topmost record-type is assumed to have SYSTEM as the owner, and the box for the SYSTEM record-type will not be drawn.) The lines in these diagrams indicate the owner-member relation of two record-types. Remember that Figure III.D.1 represents record types; Figure III.D.2 gives an example of record occurrences. (Future examples will illustrate only the record-type; the record occurrences will be implicit.)

Retrieving records is done through current owner-member combinations. The OPEN command will set SYSTEM to be the current owner of set S1 in Figure III.D.2. Since set S1 has a current owner, it is allowable (and possible) to find a member record occurrence (or member) of set S1. The method of doing this will be discussed

shortly. Once a member of S1 is found, the DMS will denote it to be the current member of S1. Other commands will allow us now to retrieve the data from the record (the item occurrences or fields).

So far, set S1 has a current owner and a current member. These are the only currency indicators that have been set. Currency indicators for set S2 are still null. (Currency indicators for "current of record type" are set by certain commands. These commands will not be used in examples presented here, and therefore changes in these currency indicators will not be mentioned in the examples. The user will learn the use of these through experience in writing routines with DML commands.)

At this point there is no way of fetching any member records of set S2 (that is, occurrences of record-type B). There is a command, though, which will set the current member of set S1 to be the current owner of S2 (SOM). So now we have found the link that attaches an occurrence of A to the related occurrences of B. Since S2 now has a current owner, then members of S2 (B) may be searched through to locate and define a current member, which in turn allows us to fetch the data from the particular occurrence of B that is current.

Example: The data base has been opened, setting a current owner for set S1. We desire the data from B4. We first search through member records of S1 to find A2 (which is automatically set to be the current member of S1). Then we set the current member of S1 to be the current owner of S2 (make A2 the current owner of S2). We now search through member records of S2 to find B4 and execute a command to get a field

from the current member of S2.

The set structure of a data base can be thought of as a network of record-types with one or more ways of getting to a record-type. Getting to a record occurrence is a matter of chaining through the structure, finding a member of one set, making it the owner of the next set, finding a member of that set and making it the owner of the next set, on down the line until the set having the desired record-type as a member is reached. Then after one more search through that last set the data is ready to be retrieved.

Figure III.D.3 shows two paths to reach record-type F. One path involves finding members and setting owners through sets S1, S3, S4, and S5. The other path is through sets S1, S2 and S7. The user need not be concerned about possible interactions between the currency indicators of these sets -- once currency indicators are set, they remain until they are changed; setting the currency indicators of one set does not affect those of another set. If the S1-S2-S7 path is chosen to fetch an occurrence of F, the user can then easily fetch an occurrence of E that is related to the current record occurrence of S2. This is done by setting the current member of S2 to be the current owner of S6 (it is already the current owner of S7).

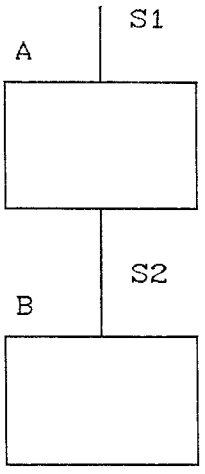


FIGURE III.D.1

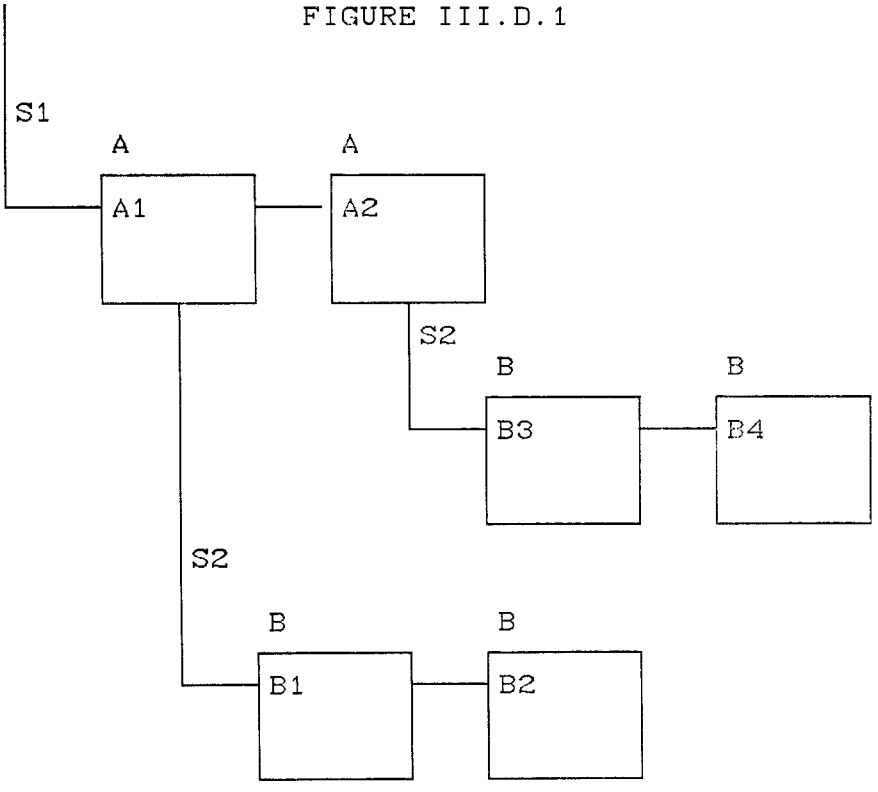


FIGURE III.D.2

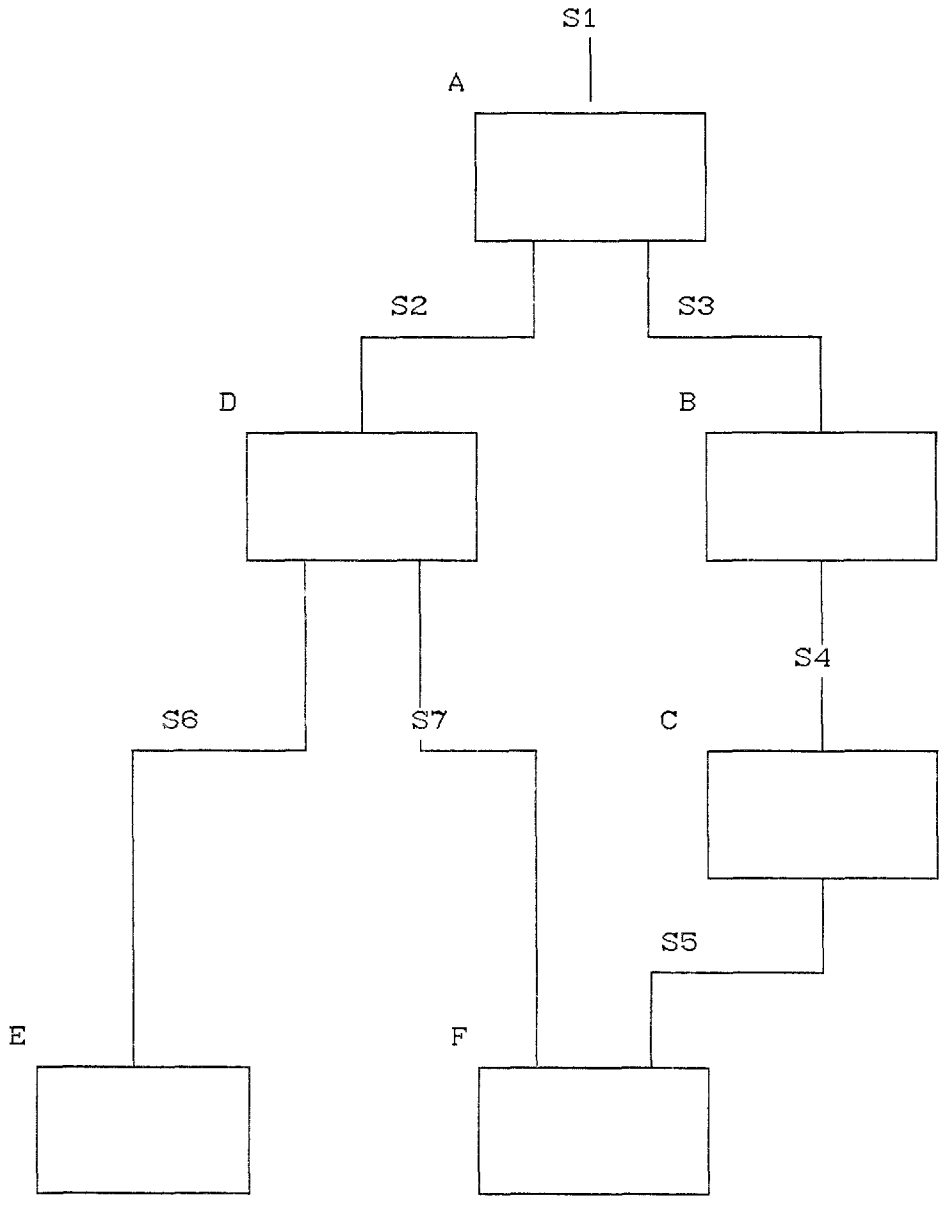


FIGURE III.D.3

The following lengthy example should clarify not only the use of the DML, but also the design of the DDL. The actual DML commands will be given along with explanations. A detailed description of each DML command is given in a later section. The sample routines are written in BASIC.

We are going to develop a system to work with a data base for a high school district containing several separate schools. We have decided beforehand what types of data will be saved: school names; teacher names and seniority; student names, number and grade point average; class titles, room and semester offered; and the grade and teacher for each class each student takes. Given this information it is a simple matter to define the record and data-items (see Figure III.D.4).

The next step, to define the sets (or how the record-types relate to one another), is based upon the type of queries that will be made of the data base. We will assume that it was decided to limit the queries to the following (refer to Figure III.D.5 while reading this list):

1. alphabetical list of all schools (handled via set S1)

2. alphabetical list of all students at a particular school (handled via sets S1 and S2A)

3. list of all students by student number at a particular school (handled via sets S1 and S2B)

4. alphabetical list of all classes offered at a particular school (handled via sets S1 and S3)

5. alphabetical list of all teachers at a particular school (handled via sets S1 and S4)

6. list of classes taught by each teacher at a particular school (handled via sets S1, S4, and S5)

7. list of classes taken by a particular student at a particular school (handled via sets S1, S2A, S6 and S7)

8. list of all students in each class at a particular school (handled via sets S1, S3, S8 and S6 [not only can a member be found from an owner, but an owner can also be found from a member])

Figure III.D.5 lists the DDL for the sets. Figure III.D.6 illustrates the set structure as defined in the DDL. The easiest method of developing the DDL is by drawing the picture first: identify each type of record, and then draw arrows between the records showing what type of record is desired to be linked from the particular record-type. Each arrow represents a set with the owner at the tail and the member at the head.

We now begin programming the desired queries. We assume the data base is already open and that the following data blocks have been defined:

Block Name	Variable List
SCHNM	S$
SCHOOL	S$
NAME	N$
NUMBER	N
AVERAGE	A
NAMET	T$
DATA	C$, R, S
DATA2	G
GRADE	F
STUDENT	P$
DATA3	N$, N, A
TITLE	L$

o1.

```
1     E0 = CALL (A0, "FFM, S1")
2     E0 = CALL (A0, "GFM, NAME,S1,SCHNM")
3     PRINT S$
4     E0 = CALL (A0, "FNM, S1")
5     IF E0 = 0 THEN 2
```

NOTES:

1 Find First Member in set S1 and make it current.

2 Get Field from Member. Get the NAME field from current member of S1 and put it in the BASIC variable S$.

4 Find Next Member of set S1 and make it current.

5 If E0 is 255, the end of the set has been reached, signifying the last school has been processed.

o2.

```
1     E0 = CALL (A0, "FMSK, S1, SCHOOL")
2     IF E0 = 0 THEN 6
3     IF E0 < 255 THEN 500
4     PRINT " SCHOOL NAME NOT FOUND "
5     GOTO 500
6     E0 = CALL (A0, "SOM, S2A ,S1")
7     E0 = CALL (A0, "GFM, NAME,S2A ,NAME")
8     E0 = CALL (A0, "GFM, NUMBER,S2A ,NUMBER")
9     E0 = CALL (A0, "GFM, GPA ,S2A ,AVERAGE")
10    PRINT N$, N, A
11    E0 = CALL (A0, "FNM, S2A ")
12    IF E0 = 0 THEN 7
```

NOTES:

1 Find Member Based on Sort Key. Search through set S1 looking for NAME equal to value of S$. The NAME field was identified as the sort key in the DDL. When found, make the record the current member of S1.

2 If E0 is non-zero, some error was encountered, most likely indicating a school with the desired name was not found.

3 Set Owner Based on Member. Make the current member of S1 the current owner of S2A. We have locked onto a particular school; we now

link to the students at that school. The first member of S2A is automatically set by this command to be the current member of S2A.

o3.

```
1     E0 = CALL (A0, "SOM, S2B ,S1")
2     E0 = CALL (A0, "FFM, S2B ")
3     E0 = CALL (A0, "GFM, NAME,S2B ,NAME")
4     E0 = CALL (A0, "GFM, NUMBER,S2B ,NUMBER")
5     E0 = CALL (A0, "GFM, GPA ,S2B ,AVERAGE")
6     PRINT N$, N, A
7     E0 = CALL (A0, "FNM, S2B")
8     IF E0 = 0 THEN 3
```

NOTES:

1 We have already set the current member of S1 from a previous query.

o4.

Similar to 3.

o5.

Similar to 3.

o6.

```
1     E0 = CALL (A0, "FMSK, S1,SCHOOL")
2     E0 = CALL (A0, "SOM, S4,S1")
3     E0 = CALL (A0, "FMSK, S4,NAMET")
4     E0 = CALL (A0, "SOM, S5,S4")
5     E0 = CALL (A0, "GETM, S5,DATA")
6     PRINT C$, R, S
7     E0 = CALL (A0, "FNM, S5")
8     IF E0 = 0 THEN 5
```

NOTES:

1 Find the school.

2 Link that school to its teachers.

3 Find the teacher's record. Here, N$ is assumed to be a string of up to 20 characters as defined in the DDL.

4 Link that teacher to the classes he teaches. Again, the first member of S5 is made the current member.

5 GET Data from Current Member. This replaces the three GFM's that would have been necessary to fetch the fields from the record. GETM will fetch all the fields from the current member of S5 and place them into the variables C$, R and S as defined by the Data Blocks.

●7.

```
1     E0 = CALL (A0, "FMSK, S1,SCHOOL")
2     E0 = CALL (A0, "SOM, S2A ,S1")
3     E0 = CALL (A0, "FMSK, S2A ,NAME")
4     E0 = CALL (A0, "GFM, NAME,S2A, NAME")
5     E0 = CALL (A0, "GFM, NUMBER, S2A ,NUMBER")
6     E0 = CALL (A0, "GFM, GPA , S2A ,AVERAGE")
7     PRINT N$, N, A
8     E0 = CALL (A0, "SOM, S6, S2A ")
9     E0 = CALL (A0, "GETM, S6, GRADE")
10    PRINT G
11    E0 = CALL (A0, "SMM, S7, S6")
12    E0 = CALL (A0, "GETO, S7, DATA")
13    PRINT C$, R, S
14    E0 = CALL (A0, "FNM, S6")
15    IF E0 = 0 THEN 9
```

NOTES:

1 Find the school.

2 Link to students.

3 Find the student.

4 Get the data.

8 Link the student to his classes.

9 Fetch the grade.

11 Link to classes.

12 Fetch class title, room, semester.

14 Find the next class the student has taken.

08.

This can be developed from the principles demonstrated in 7. The command SMM will be needed to link backwards through set S6.

We now attempt to add a new student who has taken several courses.

```
1     E0 = CALL (A0, "FMSK, S1,SCHOOL")
2     E0 = CALL (A0, "SOM, S2A ,S1")
3     E0 = CALL (A0, "SOM, S2B ,S1")
4     E0 = CALL (A0, "SOM, S3,S1")
5     E0 = CALL (A0, "FMSK, S2A ,STUDENT")
6     IF E0 = 0 THEN 10
7     E0 = CALL (A0, "CRS, STUDENT, DATA3")
8     E0 = CALL (A0, "AMS, STUDENT ,S2A")
9     E0 = CALL (A0, "AMS, STUDENT ,S2B")
10    E0 = CALL (A0, "SOM, S6, S2A")
```

Repeat the following for each of the student's classes:

```
11    E0 = CALL (A0, "FMSK, S3, TITLE")
12    IF E0 > 0 THEN 500
13    E0 = CALL (A0, "CRS, SCLASS, GRADE")
14    E0 = CALL (A0, "AMS, SCLASS, S6")
15    E0 = CALL (A0, "AMS, CLASS, S7")
```

NOTES ON ABOVE EXAMPLE:

5 See if student already exists.

7 Create a new occurrence of record-type STUDENT and store the student's name, number and grade point average.

8 Add Member to Set. Add the newly created record occurrence to set S2A, placing it in alphabetical order as indicated in the DDL.

9 Add the record occurrence to set S2B, also, placing it in numerical order.

10 Link the new student record to his classes, of which there are currently none. "S2B" could have been used in place of "S2A",

achieving the same result.

11 See if the course exists.

When removing data from the data base, the user must be very careful. Read about the DRM and the RMS commands. When a record is deleted, it is actually lost such that it can never be recovered. On the other hand, when a member is removed from a set, only the linkage between the implied owner and member is lost. This is important when a record is a member of multiple sets.


```
RECORD SCHOOL                    ALL SCHOOLS
ITEM   NAME      CHAR  10        NAME OF SCHOOL

RECORD TEACHER                   ALL TEACHERS
ITEM   NAME      CHAR  20        TEACHER NAME
ITEM   SENIORTY  INT    8        SENIORITY

RECORD STUDENT                   ALL STUDENTS
ITEM   NAME      CHAR  20        STUDENT NAME
ITEM   NUMBER    INT    8        STUDENT NUMBER
ITEM   GPA       REAL   8        GRADE POINT AVERAGE

RECORD CLASS                     ALL CLASSES OFFERED
ITEM   TITLE     CHAR  30        COURSE TITLE
ITEM   ROOM      INT    8        ROOM NUMBER
ITEM   SEMESTER  INT    8        SEMESTER OFFERED

RECORD SCLASS                    STUDENT'S CLASSES
ITEM   GRADE     INT    8        GRADE RECEIVED
```

FIGURE III.D.4

```
SET     S1          AUTO 1:N      Schools sorted by name
                    SORT NAME
OWNER   SYSTEM
MEMBER  SCHOOL


SET     S2A         MAN  1:N      Students sorted by name
                    SORT NAME
OWNER   SCHOOL
MEMBER  STUDENT


SET     S2B         MAN  1:N      Students sorted by number
                    SORT NUMBER
OWNER   SCHOOL
MEMBER  STUDENT


SET     S3          MAN  1:N      Classes offered by a school
                    SORT TITLE
OWNER   SCHOOL
MEMBER  CLASS


SET     S4          MAN  1:N      Teachers in a school
                    SORT NAME
OWNER   SCHOOL
MEMBER  TEACHER


SET     S5          MAN  1:N      Teachers teach classes
                    FIFO
OWNER   TEACHER
MEMBER  CLASS


SET     S6          MAN  1:N      Students take classes
                    FIFO
OWNER   STUDENT
MEMBER  SCLASS


SET     S7          MAN  1:N      Classes consist of students
                    IMMAT
OWNER   CLASS
MEMBER  SCLASS
```

FIGURE III.D.5

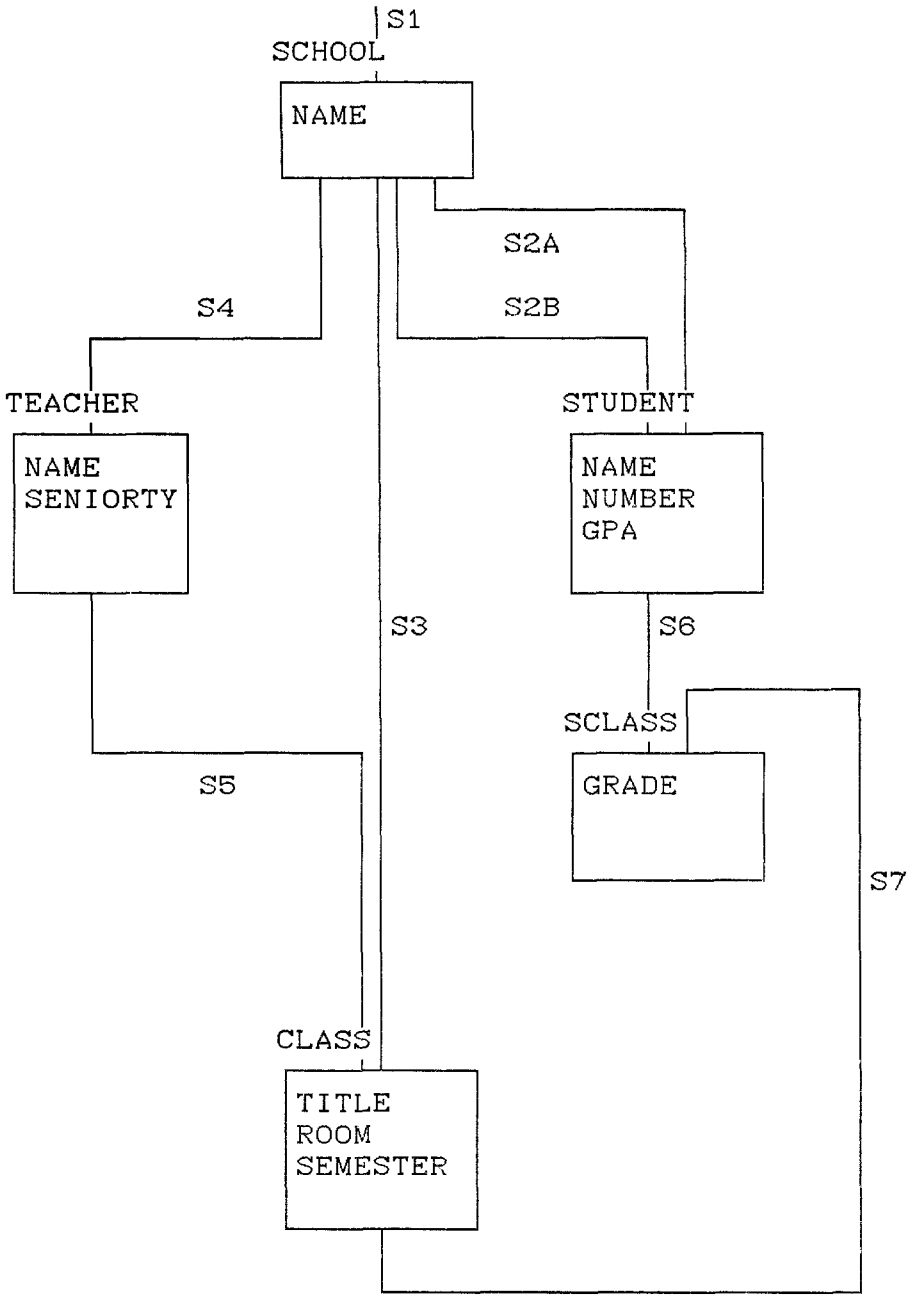


FIGURE III.D.6

2. Calling Procedures

MDBS.DMS routines can be called from either machine language routines or from higher level languages such as BASIC, FORTRAN, COBOL, PASCAL, PL/I, etc. While the machine language calls result in a faster more efficient usage of the DMS package, calling from a higher level language simplifies programmer tasks and results in a quick and easily programmed data base application.

a. Calls from Machine Language Routines.

Machine language callable forms of MDBS.DMS use calling conventions that are a function of the CPU. These are discussed in detail in the appropriate system specific manual.

b. Calls From Higher Level Languages

A call to a DML routine from a higher level language (such as BASIC) will look (generically) like:

E0 = CALL (A, "routine name, arguments", host language arguments)

where:

A	A DMS entry point address
Routine name	Name of a DML routine
Arguments	A list of arguments, separated by commas, giving item, record, set or data block names as required by the DML routines.
Host language arguments	Host language variables separated by commas. These are used only in the

DEFINE and EXTEND DML commands.

E0 — Represents a program variable where a status value is returned.

The exact form of host language calls on your system is given in the system specific manual.

A powerful feature of the Data Management System is the usage of data blocks. A data block is a named collection of host language variables. A given variable may participate in more than one data block.

As an illustration, consider the following record type that may appear in an order processing system:

```
RECORD  CUSTOMER
ITEM    NAME     CHAR  16
ITEM    STREET   CHAR  16
ITEM    CITY     CHAR  16
ITEM    STATE    CHAR   2
ITEM    ZIPCODE  CHAR   5
ITEM    CREDLIM  REAL   8
ITEM    LASTSIZE REAL   8
```

We wish to store the customer's name, address, credit limit, and size of the last purchase.

A BASIC user can issue the following call to DMS:

```
E0 = CALL (A1, "DEFINE, DETAIL", N$, S$, C$, T$, Z$, C, S)
```

which will define a data block named DETAIL having variables N$, S$, C$, T$, Z$, C and S. Whenever the data block DETAIL is referred after this definition, its variables will either receive (supply) values from (to) the data base. For example, to get data from the current customer record we would use:


```
E0 = CALL (A0, "GETR, CUSTOMER, DETAIL")
```

and this has the effect of retrieving the contents of the current customer record and saving the contents of each customer field in the associated host language variable. Likewise, a call to create a record and store data:

```
E0 = CALL (A0,"CRS, CUSTOMER, DETAIL")
```

will store the contents of each variable in the corresponding record field.

In many cases it is convenient to let the MDBS.DMS system select a default data block name by omitting the data block name parameter in the call statement:

```
E0 = CALL (A0,"CRS, CUSTOMER")
```

In this case, a data block name of CUSTOMER will be used. If the command specified is a record-oriented command (CRS, GETx, PUTx), the record name is used as the default data block name. If the command is an item-oriented command (GFx, SFx), the item name is used as a the default data block name.

3. Data Management System Routines

DMS ERRORS

Error	Explanation
01	Data Base not open
02	Invalid set-type
03	Invalid record-type
04	Invalid item-type for this record-type
05	Invalid owner-type for this set-type
06	Invalid member-type for this set-type
07	Invalid data base key
08	No current owner of set-type
09	No current member of set-type
10	No current of record-type
11	Record already member of set
12	Record not member of set
13	Depending on item too large or negative
14	Data Base already open
15	Data Base not closed previously
16	No current of run unit
17	No more space in Data Base
18	Set not sorted
19	Depending-on item not binary with size 2 or it is a replicated item
20	A record type cannot have more than 1 variable length item
24	Duplicate name specified
26	Sole owner/member may not be deleted
27	Depending item or sort key may not be deleted
28	Invalid set characteristics
29	System-owned set cannot have other owners
32	Data Base opened for read access only
33	Record size too large
34	Invalid number
36	Improper password
40	Maximum value of depending on item is 32767
41	Binary number too large
89	Variable length inconsistency
90	No such DMS routine
91	Insufficient room in memory
92	Incorrect number of arguments
93	Duplicate data-block name
94	Block name not found
95	Invalid data block name
96	Invalid number of arguments
97	Cannot read from data base files
98	Record occurrence(s) lost
99	Catastrophe
100	User may not read this record
101	User may not write this record

102	User may not read this item
103	User may not write this item
104	Disk in wrong drive
105	Disk read or write error
106	Cannot expand file for new page
107	File not present
108	User may not read this set
109	User may not write this set
110	Access levels inconsistent
111	Syntax error in command line
255	End-of-set or end of specification

Add Current of run unit to Set — ACS

E0 = CALL (A0, "ACS, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current of run unit(input)
current owner of set-type(input)
current member of set-type(input)

current member of set-type(output)

Description:

The current record of the run unit is added to the set occurrence identified by the current owner of that set.

The position of the new member in the set is determined by the ordering criteria given in the DDL Set Description for the given set-type. If the set has been defined as a N:1 or N:M set the position of the current owner with respect to other owners of the member record occurrence is determined similarly.

The new member becomes the current member of the given set-type.

Errors:

01. data base not open
02. invalid set-type
06. invalid member type for this set-type
08. no current owner of set-type
11. record already member of set
16. no current of run unit
17. no more space in Data Base
90. no such DMS routine
99. catastrophe
105. disk read or write error
106. cannot expand file for new page
109. user may not write this set

Add Member to Set AMS

E0 = CALL (A0, "AMS, record-type, set-type")

Arguments:

record-type(input)
set-type(input)

Currency Indicators Involved:

current of record-type(input)
current owner of set-type(input)
current member of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The current record of the given record-type is added to the set occurrence identified by the current owner of that set.

The position of the new member in the set is determined by the ordering criteria given in the DDL Set Description for the given set-type. If the set has been defined as a N:M or a N:1 set the position of the current owner with respect to other owners of the member record occurrence is determined similarly.

The new member becomes the current member of the given set-type.

Errors:

01. data base not open
02. invalid set-type
03. invalid record-type
06. invalid member type for this set-type
08. no current owner of set-type
10. no current of record-type
11. record already member of set
17. no more space in Data Base
90. no such DMS routine
99. catastrophe

104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page
107. file not present
109. user may not write this set

Check Current of run unit Type CCT

E0 = CALL (A0, "CCT, record-type")

Arguments:

record-type(input)

Currency Indicators Involved:

current of run unit(input)

Description:

The record-type of the current record of the run unit is compared to the given record-type. If the current type is equal to the given record-type, E0 = 0. If the current type is not equal to the given record-type, E0 = 3.

Errors:

01.	data base not open
03.	invalid record-type
16.	no current of run unit
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present

CLOSE the data base CLOSE

E0 = CALL (A0, "CLOSE")

Arguments:

none

Currency Indicators Involved:

not applicable

Description:

This routine must be the last DML command executed in any program that uses the data base. Otherwise, the data base file will be inconsistent. All buffers are rewritten, and the data base is properly closed.

Errors:

90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present

Check current Member Type — CMT

E0 = CALL (A0, "CMT, record-type, set-type")

Arguments:

record-type(input)
set-type(input)

Currency Indicators Involved:

current member of set-type(input)

Description:

The record-type of the current member of the given set-type is compared to the given record-type. If the current member-type is equal to the given record-type, E0 = 0. If the current member-type is not equal to the given record-type, E0 = 3.

Errors:

01. data base not open
02. invalid set-type
03. invalid record-type
09. no current member of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Check current Owner Type — COT

E0 = CALL (A0, "COT, record-type, set-type")

Arguments:

record-type(input)
set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

Description:

The record-type of the current owner of the given set-type is compared to the given record-type. If the current owner-type is equal to the given record-type, E0 = 0. If the current owner-type is not equal to the given record-type, E0 = 3.

Errors:

01. data base not open
02. invalid set-type
03. invalid record-type
08. no current owner of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

95. invalid data block name
96. invalid number of arguments
99. catastrophe
101. user may not write this record
103. user may not write this item
105. disk read or write error
106. cannot expand file for new page
107. file not present

DEFINE a data block DEFINE

E0 = CALL (A1, "DEFINE, data-block", hlv)

Arguments:

data-block name (input)
hlv - host language variables (input)

Currency Indicators Involved:

none

Description:

A data block with the user specified name is created and the indicated host language variables are associated with the data block.

Errors:

01. data base not open
90. no such DMS routine
91. insufficient room in memory
92. incorrect number of arguments
93. duplicate data block name
95. invalid data block name
96. invalid number of arguments
99. catastrophe

Notes:

1. A1 is the address of the entry to the DMS for the DEFINE and EXTEND DML commands.
2. The hlv list contains at least one variable name (constants are not allowed) and, if there is more than one variable, the variables are separated by commas.
3. A given host language variable can appear in more than one data block.
4. Any host language variable can appear more than once in a data block.
5. DEFINE can be called before or after an OPEN.

Delete Record based on Current of run unit DRC

E0 = CALL (A0, "DRC")

Arguments:

Currency Indicators Involved:

current of run unit(input)

all indicators referencing the specified record (output)

Description:

The record identified by the current of the run unit is logically and physically deleted from the data base.

For all sets of which the record to be deleted is an owner, the set occurrence is deleted.

For all sets of which the record to be deleted is a member, the record is removed from the set, i.e., the previous member is linked to the next member relative to the deleted member.

For all set-types of which the deleted record was the current owner, the current owner and current member currency indicators are set to null.

For all set-types of which the deleted record was the current member, the currency indicator of the member is set to null. (Note that it is possible to reach the end of the set this way without it being indicated via E0).

If the deleted record was the current of its record-type, the currency indicator of that record-type is set to null.

Additionally, the currency indicator of the run unit is set to null.

Errors:

01. data base not open
02. invalid set-type

16. no current of run unit
90. no such DMS routine
99. catastrophe
101. user may not write this record
104. disk in wrong drive
105. disk read or write error
107. file not present

Delete Record based on current Member DRM

E0 = CALL (A0, "DRM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current member of set-type(input)

all indicators referencing the specified record (output)

Description:

The record identified by the current member of the set-type is logically and physically deleted from the data base.

For all sets of which the record to be deleted is an owner, the set occurrence is deleted.

For all sets of which the record to be deleted is a member, the record is removed from the set, i.e., the previous member is linked to the next member relative to the deleted member.

For all set-types of which the deleted record was the current owner, the current owner and current member currency indicators are set to null.

For all set-types of which the deleted record was the current member, the currency indicator of the member is set to null. (Note that it is possible to reach the end of the set this way without it being indicated via E0).

If the deleted record was the current of its record-type, the currency indicator of that record-type is set to null.

If the deleted record was the current of the run unit, the current of the run unit is set to null.

Errors:

01.	data base not open
02.	invalid set-type
09.	no current member of set-type
90.	no such DMS routine
99.	catastrophe
101.	user may not write this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set

Delete Record based on current Owner DRO

E0 = CALL (A0, "DRO, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

all indicators referencing the specified record (output)

Description:

The record identified by the current owner of the set-type is logically and physically deleted from the data base.

For all sets of which the record to be deleted is an owner, the set occurrence is deleted.

For all sets of which the record to be deleted is a member, the record is removed from the set, i.e., the previous member is linked to the next member relative to the deleted member.

For all set-types of which the deleted record was the current owner, the current owner and current member currency indicators are set to null.

For all set-types of which the deleted record was the current member, the currency indicator of the member is set to null. (Note that it is possible to reach the end of the set this way without it being indicated via E0).

If the deleted record was the current of its record-type, the current of that record-type is set to null.

If the deleted record was the current of the run unit, the currency indicator of the current of run unit is set to null.

Errors:

01. data base not open

02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
99. catastrophe
101. user may not write this record
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Delete Record based on current Record DRR

E0 = CALL (A0, "DRR, record-type")

Arguments:

record-type(input)

Currency Indicators Involved:

current of record-type(input)

all indicators referencing the specified record (output)

Description:

The record identified by the current of the record-type is logically and physically deleted from the data base.

For all sets of which the record to be deleted is an owner, the set occurrence is deleted.

For all sets of which the record to be deleted is a member, the record is removed from the set, i.e., the previous member is linked to the next member relative to the deleted member.

For all set-types of which the deleted record was the current owner, the current owner and current member currency indicators are set to null.

For all set-types of which the deleted record was the current member, the currency indicator of the member is set to null. (Note that it is possible to reach the end of the set this way without it being indicated via E0).

If the deleted record was the current of its record-type, the currency indicator of that record-type is set to null.

If the deleted record was the current of the run unit, the currency indicator of the current of run unit is set to null.

Errors:

01.	data base not open
03.	invalid record-type
10.	no current of record-type
90.	no such DMS routine
99.	catastrophe
101.	user may not write this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present

EXTEND a data block EXTEND

E0 = CALL (A1, "EXTEND, data-block", hlv)

Arguments:

data block name (input)
hlv - host language variables (input)

Currency Indicators Involved:

none

Description:

This routine adds more host language variables to an already defined data-block. This is useful if the size of a host language line of text is too limited.

Errors:

90. no such DMS routine
91. insufficient room in memory
92. incorrect number of arguments
94. data block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe

Notes:

1. An EXTEND call for a given block must follow either another EXTEND call for that block or the DEFINE call for that block. Other EXTEND or DEFINE calls cannot be intermixed.

2. The call:
   E0 = CALL (A1, "DEFINE, RX", A, B, C, D, E, F)
   is equivalent to:
   E0 = CALL (A1, "DEFINE, RX", A,B,C,D)
   E0 = CALL (A1, "EXTEND, RX", E, F)

3. The hlv list must contain at least one variable name (constants are not allowed) and, if there is more than one variable, the variables are separated by commas.

4. A given host language variable can appear in more than one data block.

5. A host language variable can appear more than once in a data block.

6. EXTEND can be called before or after an OPEN.

Find First Member FFM

E0 = CALL (A0, "FFM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The first member of the given set-type is made the current member of that set-type.

If the set contains no members, E0 is set to 255.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find First Owner FFO

E0 = CALL (A0, "FFO, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current member of set-type(input)

current owner of set-type(output)
current of run unit(output)

Description:

The first owner of the given set-type is made the current owner of that set-type.

Errors:

01. data base not open
02. invalid set-type
09. no current member of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

FIND Member FINDM

E0 = CALL (A0, "FINDM, item-type, set-type, data-block")

Arguments:

item-type
set-type(input)
data-block(input)

Currency Indicators Involved:

current owner of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The current set is searched for the logically first member in which the item-type specified has a value equal to the value given in the data-block. If such a member is found, it is made the current member of the given set-type. If such a member is not found, E0 is set to 255.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
94. block name not found
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

255. end-of-set

FIND Owner FINDO

E0 = CALL (A0, "FINDO, item-type, set-type, data-block")

Arguments:

item-type
set-type(input)
data-block(input)

Currency Indicators Involved:

current member of set-type(input)

current owner of set-type(output)
current of run unit(output)

Description:

The current set is searched for the logically first owner in which the item-type specified has a value equal to the value given in the data-block. If such an owner is found, it is made the current owner of the given set-type. If such a owner is not found, E0 is set to 255.

Errors:

01.	data base not open
02.	invalid set-type
09.	no current member of set-type
90.	no such DMS routine
94.	block name not found
96.	invalid number of arguments
99.	catastrophe
100.	user may not read this record
102.	user may not read this item
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find Last Member FLM

E0 = CALL (A0, "FLM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The last member of the given set-type is made the current member of that set-type.

If the set contains no members, E0 is set to 255.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find Last Owner FLO

E0 = CALL (A0, "FLO, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current member of set-type(input)

current owner of set-type(output)
current of run unit(output)

Description:

The last owner of the given set-type is made the current owner of that set-type.

Errors:

01.	data base not open
02.	invalid set-type
09.	no current member of set-type
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set

Find Member based on Sort Key FMSK

E0 = CALL (A0, "FMSK, set-type, data-block")

Arguments:

set-type(input)
data-block(input)

Currency Indicators Involved:

current owner of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The current set is searched for the logically first member with a sort key value equal to the value given in the data-block. If such a member is found, it is made the current member of the given set-type and the current record of the run unit. If such a member is not found, E0 is set to 255 and the current member of the set-type is set to the record logically prior to the requested record. Note that the current of run unit is not altered when error 255 is returned.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
90.	no such DMS routine
94.	block name not found
96.	invalid number of arguments
99.	catastrophe
100.	user may not read this record
102.	user may not read this item
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find Next Member FNM

E0 = CALL (A0, "FNM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The logically next member of the given set-type is made the current member of the given set-type.

If there is no next member (the current member is the last, or the set is empty) E0 is set to 255 and the currency indicator for the current member of the set is not affected.

If there is no current member of the given set-type, then this is the same as FFM.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find Next Owner FNO

E0 = CALL (A0, "FNO, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

current member of set-type(input)

current owner of set-type(output)
current of run unit(output)

Description:

The logically next owner of the given set-type is made the current owner of the given set-type.

If there is no next owner (the current owner is the last) E0 is set to 255 and the currency indicator for the current owner of the set is not affected.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
09.	no current member of set-type
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find Owner based on Sort Key — FOSK

E0 = CALL (A0, "FOSK, set-type, data-block")

Arguments:

set-type(input)
data-block(input)

Currency Indicators Involved:

current member of set-type(input)

current owner of set-type(output)
current of run unit(output)

Description:

The current set is searched for the logically first owner with a sort key value equal to the value given in the data-block. If such an owner is found, it is made the current owner of the given set-type and the current record of the run unit. If such a owner is not found, E0 is set to 255 and the current member of the set-type is set to the record logically prior to the requested record. Note that the current run unit is not altered when error 255 is returned.

Errors:

01.	data base not open
02.	invalid set-type
09.	no current member of set-type
90.	no such DMS routine
94.	block name not found
96.	invalid number of arguments
99.	catastrophe
100.	user may not read this record
102.	user may not read this item
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

Find Previous Member FPM

E0 = CALL (A0, "FPM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

current member of set-type(output)
current of run unit(output)

Description:

The logically previous member of the given set-type is made the current member of the given set-type.

If there is no previous member (the current member is the first, or the set is empty) E0 is set to 255 and the currency indicator for the current member of the set is not affected.

If there is no current member of the given set-type, then this is the same as FLM.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

255. end-of-set

Find Previous Owner FPO

EO = CALL (AO, "FPO, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

current owner of set-type(output)
current of run unit(output)

Description:

The logically previous owner of the given set-type is made the current owner of the given set-type.

If there is no previous owner (the current owner is the first) EO is set to 255 and the currency indicator for the current owner of the set is not affected.

Errors:

01.	data base not open
C2.	invalid set-type
08.	no current owner of set-type
09.	no current member of set-type
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
255.	end-of-set

GET data from Current of run unit GETC

E0 = CALL (A0, "GETC, data-block")

Arguments:

data-block(output)

Currency Indicators Involved:

current of run unit(input)

Description:

The value of all items associated with the current record of the run unit are returned in the data-block. The values are returned in the same order as the items in the DDL for the given record-type.

There is no check made that variables in the data-block are of the right type or size.

Errors:

01. data base not open
16. no current of run unit
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present

GET data from current Member GETM

EO = CALL (AO, "GETM, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

Description:

The value of all items associated with the current member of the set-type are returned in the data-block. The values are returned in the same order as the items in the DDL for the given record-type.

There is no check made that the variables in the data-block are of the correct type or size. These checks are solely the responsibility of the applications programmer.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
09. no current member of set-type
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

GET data from current Owner GETO

E0 = CALL (A0, "GETO, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)

Description:

The value of all items associated with the current owner of the set-type are returned in the data-block. The values are returned in the same order as the items in the DDL for the given record-type.

There is no check made that the variables in the data-block are of the correct type or size. These checks are solely the responsibility of the applications programmer.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

GET data from current Record GETR

E0 = CALL (A0, "GETR, record-type, data-block")

Arguments:

record-type(input)

data-block(output)

Currency Indicators Involved:

current of record-type(input)

Description:

The value of all items associated with the current of the specified record type are returned in the data-block. The values are returned in the same order as the items in the DDL for the given record-type.

There is no check made that the variables in the data-block are of the correct type or size. These checks are solely the responsibility of the applications programmer.

Errors:

01. data base not open
03. invalid record-type
10. no current of record-type
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present

Get Field from Current of run unit GFC

```
E0 = CALL (A0, "GFC, item-type, data-block")
```

Arguments:

item-type(input)

data-block(output)

Currency Indicators Involved:

current of run unit(input)

Description:

The value of the given item-type in the record identified by the current record of the run unit is returned in the data-block.

There is no check made that the variable in the data-block is of the correct type or size. These checks are solely the responsibility of the applications programmer.

If the item-type is a depending item, the depended on item is used to determine the number of instances of the item to return.

Errors:

01. data base not open
02. invalid set-type
04. invalid item-type for this record-type
13. depended on item too large or negative
16. no current of run unit
90. no such DMS routine
94. block name not found
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present

Get Field from current Member GFM

E0 = CALL (A0, "GFM, item-type, set-type, data-block")

Arguments:

item-type(input)
set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

Description:

The value of the given item-type in the record identified by the current member of the specified set-type is returned in the data-block.

There is no check made that the variable in the data-block is of the correct type or size. These checks are solely the responsibility of the applications programmer.

If the item-type is a depending item, the depended on item is used to determine the number of instances of the item to return.

Errors:

01. data base not open
02. invalid set-type
04. invalid item-type for this record-type
08. no current owner of set-type
09. no current member of set-type
13. depended on item too large or negative
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Get Field from current Owner GFO

E0 = CALL (A0, "GFO, item-type, set-type, data-block")

Arguments:

item-type(input)
set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)

Description:

The value of the given item-type in the record identified by the current owner of the specified set-type is returned in the data-block.

There is no check made that the variable in the data-block is of the correct type or size. These checks are solely the responsibility of the applications programmer.

If the item-type is a depending item, the depended on item is used to determine the number of instances of the item to return.

Errors:

01. data base not open
02. invalid set-type
04. invalid item-type for this record-type
08. no current owner of set-type
13. depended on item too large or negative
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Get Field from current Record GFR

E0 = CALL (A0, "GFR, item-type, record-type, data-block")

Arguments:

item-type(input)
record-type(input)

data-block(output)

Currency Indicators Involved:

current of record-type(input)

Description:

The value of the given item-type in the record identified by the current record of the specified record-type is returned in the data-block.

There is no check made that the variable in the data-block is of the correct type or size. These checks are solely the responsibility of the applications programmer.

If the item-type is a depending item, the depended on item is used to determine the number of instances of the item to return.

Errors:

01. data base not open
03. invalid record-type
04. invalid item-type for this record-type
10. no current of record-type
13. depended on item too large or negative
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
102. user may not read this item
104. disk in wrong drive
105. disk read or write error
107. file not present

Get Member Count GMC

E0 = CALL (A0, "GMC, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)

Description:

This routine returns the number of member record occurrences owned by the current owner of the set-type. The data-block must contain two integer variables. For illustration purposes, call them I and J. The total number of occurrences is computed by the user with the formula:

Total Number = 32768 * I + J

Normally, I will be zero.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
94. block name not found
96. invalid number of arguments
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Get Owner Count GOC

E0 = CALL (A0, "GOC, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current member of set-type(input)

Description:

This routine returns the number of owner record occurrences owned by the current member of the set-type. The data-block must contain two integer variables. For illustration purposes, call them I and J. The total number of occurrences is computed by the user with the formula:

Total Number = 32768 * I + J

Normally, I will be zero.

Errors:

01. data base not open
02. invalid set-type
09. no current member of set-type
90. no such DMS routine
94. block name not found
96. invalid number of arguments
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Get record-Type of Current of run unit GTC

E0 = CALL (A0, "GTC, data-block")

Arguments:

data-block(output)

Currency Indicators Involved:

current of run unit(input)

Description:

The record-type name of the record identified by the current record of the run unit is returned in the first data-block variable.

The record-type name is in character format, and is padded with trailing blanks.

Errors:

01. data base not open
16. no current of run unit
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Get record-Type of current Member GTM

E0 = CALL (A0, "GTM, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

Description:

The record-type name of the record identified by the current member of the set-type is returned in the first data-block variable.

The record-type name is in character format, and is padded with trailing blanks.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
09. no current member of set-type
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
100. user may not read this record
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

Get record-Type of current Owner GTO

E0 = CALL (A0, "GTO, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)

Description:

The record-type name of the record identified by the current owner of the set-type is returned in the first data-block variable.

The record-type name is in character format, and is padded with trailing blanks.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

OPEN data base OPEN

E0 = CALL (A0, "OPEN, data-block")

Arguments:

data-block (input)

Currency indicators involved:

all

Description:

This subroutine must be called before any other of the subroutines in the DMS are called (except DEFINE and EXTEND).

Table initialization is taken care of by this subroutine.

All sets which have been defined in the DDL as having SYSTEM as their owner have their current owner set to SYSTEM. The current of run unit is set to SYSTEM. All other currency indicators are set to null.

The data block must contain the following:

a. Data base file name
   This must be a character string containing a fully qualified file name as specified in the DDL.
b. User's name
   This must be an upper case character string containing the user's name as declared in the PASSWORDS section of the DDL.
c. Password
   This must be an upper case character string containing the user's password as declared in the PASSWORDS section of the DDL.
d. Read/write status
   This must be an upper case character string containing "MOD" if the user wishes to write and read from the data base. Any other string will put the data base in a read-only mode.

Errors:

14. data base already open
15. data base not closed previously
36. improper password
90. no such DMS routine
91. insufficient room in memory
92. incorrect number of arguments
94. block name not found
95. invalid data block name
96. invalid number of arguments
97. cannot read from data base files
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present

PUT data into Current of run unit — PUTC

E0 = CALL (A0, "PUTC, data-block")

Arguments:

data-block(output)

Currency Indicators Involved:

current of run unit(input)

Description:

The data-block specified is stored in the record specified by the current record of the run unit. The data-block is assumed to be in the same order and alignment as the items in the DDL for the given record-type.

There is no check made that data-block is of the right type, alignment or size.

If any item-type in the given record-type is a depending item, the location of the depended on item is assumed to contain the correct value to determine the number of instances of the depending on item to store.

Errors:

01. data base not open
16. no current of run unit
89. variable length inconsistency
90. no such DMS routine
94. block name not found
95. invalid data block name
99. catastrophe
101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page
107. file not present

PUT data into current Member PUTM

EO = CALL (AO, "PUTM, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

Description:

The data-block specified is stored in the record indicated by the current member of the specified set-type. The data-block is assumed to be in the same order and alignment as the items in the DDL for the given record-type.

There is no check made that data-block is of the right type, alignment or size.

If any item-type in the given record-type is a depending item, the location of the depended on item is assumed to contain the correct value to determine the number of instances of the depending on item to store.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
09. no current member of set-type
89. variable length inconsistency
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page

107. file not present
108. user may not read this set

PUT data into current Owner PUTO

E0 = CALL (A0, "PUTO, set-type, data-block")

Arguments:

set-type(input)

data-block(output)

Currency Indicators Involved:

current owner of set-type(input)

Description:

The data-block specified is stored in the record indicated by the current record of the specified set-type. The data-block is assumed to be in the same order and alignment as the items in the DDL for the given record-type.

There is no check made that data-block is of the right type, alignment or size.

If any item-type in the given record-type is a depending item, the location of the depended on item is assumed to contain the correct value to determine the number of instances of the depending on item to store.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
99. catastrophe
89. variable length inconsistency
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
90. no such DMS routine
99. catastrophe
101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error

106. cannot expand file for new page
107. file not present
108. user may not read this set

PUT data into current Record PUTR

E0 = CALL (A0, "PUTR, record-type, data-block")

Arguments:

record-type(input)

data-block(output)

Currency Indicators Involved:

current of record-type(input)

Description:

The data-block specified is stored in the record indicated by the current of the specified record-type. The data-block is assumed to be in the same order and alignment as the items in the DDL for the given record-type.

There is no check made that data-block is of the right type, alignment or size.

If any item-type in the given record-type is a depending item, the location of the depended on item is assumed to contain the correct value to determine the number of instances of the depending on item to store.

Errors:

01. data base not open
03. invalid record-type
10. no current of record-type
89. variable length inconsistency
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe
101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page
107. file not present

Remove current Member from Set RMS

E0 = CALL (A0, "RMS, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

current member of set-type(output)

Description:

The current member of the given set-type is logically removed from the set, i.e., the previous member is linked to the next member, and the current member is no longer associated with the current owner of the set-type.

The currency indicator of the member of the given set-type becomes null. (Note that it is possible to reach the end of the set this way without it being indicated via E0).

Note that the record is only removed from the given set; it is not deleted from the data base.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
09. no current member of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set
109. user may not write this set

Remove all Set Members RSM

E0 = CALL (A0, "RSM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

current member of set-type(output)

Description:

The current set is set to have no members, i.e., all members of the set are logically removed from the set.

The currency indicator of the member of the given set-type becomes null.

Note that the records removed from the set are not deleted from the data base, nor is the owner of the given set-type deleted from the data base.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set
109. user may not write this set

Set Current of run unit based on Member SCM

E0 = CALL (A0, "SCM, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type (input)
current member of set-type (input)

current of run unit(output)

Description:

The record identified by the current member of the specified set-type is set to be the current record of the run unit.

Errors:

01. data base not open
02. invalid set-type
08. no current owner of set-type
09. no current member of set-type
90. no such DMS routine
99. catastrophe
108. user may not read this set

Set Current of run unit based on Owner SCO

E0 = CALL (A0, "SCO, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

current of run unit(output)

Description:

The record identified by the current owner of the specified set-type is set to be the current record of the run unit.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
90.	no such DMS routine
99.	catastrophe
108.	user may not read this set

Set Current of run unit based on Record SCR

EO = CALL (AO, "SCR, record-type")

Arguments:

record-type(input)

Currency Indicators Involved:

current record of record type(input)

current of run unit(output)

Description:

The record identified by the current of the specified record-type is set to be the current record of the run unit.

Errors:

01.	data base not open
03.	invalid record-type
10.	no current of record-type
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record

Set Field in Current of run unit SFC

E0 = CALL (A0, "SFC, item-type, data-block")

Arguments:

item-type(input)
data-block(input)

Currency Indicators Involved:

current of run unit(input)

Description:

The value of the variable in the data-block is stored in the given item-type of the record identified by the current record of the run unit.

There is no check made that the variable is of the right type or size.

If the item-type is a depending item, the depended on item value in the record is used to determine the number of instances of the given item-type to store.

Errors:

01. data base not open
04. invalid item-type for this record-type
16. no current of run unit
13. depended on item too large or negative
40. maximum value of depending on item is 32768
90. no such DMS routine
94. block name not found
96. invalid number of arguments
99. catastrophe
101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page
107. file not present
108. user may not read this set

Set Field in current Member SFM

E0 = CALL (A0, "SFM, item-type, set-type, data-block")

Arguments:

item-type(input)
set-type(input)
data-block(input)

Currency Indicators Involved:

current owner of set-type(input)
current member of set-type(input)

Description:

The value of the variable in the data-block is stored in the given item-type of the record identified by the current member of the specified set-type.

There is no check made that the variable is of the right type or size.

If the item-type is a depending item, the depended on item value in the record is used to determine the number of instances of the given item-type to store.

Errors:

01. data base not open
02. invalid set-type
04. invalid item-type for this record-type
08. no current owner of set-type
09. no current member of set-type
13. depended on item too large or negative
40. maximum value of depending on item is 32768
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page
107. file not present

101. user may not write this record
103. user may not write this item
104. disk in wrong drive
105. disk read or write error
106. cannot expand file for new page
107. file not present
108. user may not read this set

Set Field in current Record SFR

E0 = CALL (A0, "SFR, item-type, record-type, data-block")

Arguments:

item-type(input)
record-type(input)
data-block(input)

Currency Indicators Involved:

current record of record-type(input)

Description:

The value of the variable in the data-block is stored in the given item-type of the record identified by the current record of the run unit.

There is no check made that the variable is of the right type or size.

If the item-type is a depending item, the depended on item value in the record is used to determine the number of instances of the given item-type to store.

Errors:

01.	data base not open
04.	invalid item-type for this record-type
10.	no current of record-type
13.	depended on item too large or negative
40.	maximum value of depending on item is 32768
90.	no such DMS routine
94.	block name not found
95.	invalid data block name
96.	invalid number of arguments
101.	user may not write this record
103.	user may not write this item
104.	disk in wrong drive
105.	disk read or write error
106.	cannot expand file for new page
107.	file not present
108.	user may not read this set

Set Member based on Current of run unit SMC

E0 = CALL (A0, "SMC, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current of run unit(input)

current owner of set-type(output)
current member of set-type(output)

Description:

The record identified by the current record of the run unit becomes the current member of the given set-type. The logically first owner associated with the new current member becomes the new current owner of the given set-type.

Errors:

01.	data base not open
02.	invalid set-type
06.	invalid member type for this set-type
08.	no current owner of set-type
16.	no current of run unit
12.	record not member of set
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
109.	user may not write this set
255.	end-of-set

Set current Member based on current Member SMM

E0 = CALL (A0, "SMM, set-type-1, set-type-2")

Arguments:

set-type-1(input)
set-type-2(input)

Currency Indicators Involved:

current owner of set-type-2(input)
current member of set-type-2(input)

current owner of set-type-1(output)
current member of set-type-1(output)
current of run unit(output)

Description:

The record identified by the current member of the second set-type specified becomes the current member of the first set-type. The logically first owner associated with the new current member becomes the new current owner of the given set-type.

Errors:

01.	data base not open
02.	invalid set-type
06.	invalid member type for this set-type
08.	no current owner of set-type
09.	no current member of set-type
12.	record not member of set
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
109.	user may not write this set
255.	end-of-set

Set current Member based on current Owner SMO

E0 = CALL (A0, "SMO, set-type-1, set-type-2")

Arguments:

set-type-1(input)
set-type-2(input)

Currency Indicators Involved:

current owner of set-type-2(input)
current member of set-type-2(input)

current owner of set-type-1(output)
current member of set-type-1(output)
current of run unit(output)

Description:

The record identified by the current owner of the second set-type specified becomes the current member of the first set-type. The logically first owner associated with the new current member becomes the new current owner of the given set-type.

Errors:

01.	data base not open
02.	invalid set-type
06.	invalid member type for this set-type
08.	no current owner of set-type
12.	record not member of set
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
109.	user may not write this set
255.	end-of-set

Set current Member based on current Record SMR

E0 = CALL (A0, "SMR, record-type, set-type")

Arguments:

record-type(input)
set-type(input)

Currency Indicators Involved:

current record of record-type(input)

current owner of set-type(output)
current member of set-type(output)
current of run unit(output)

Description:

The record identified by the current of the specified record-type becomes the current member of the given set-type. The logically first owner associated with the new current member becomes the new current owner of the given set-type.

Errors:

01.	data base not open
02.	invalid set-type
03.	invalid record-type
06.	invalid member type for this set-type
08.	no current owner of set-type
10.	no current of record-type
12.	record not member of set
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
109.	user may not write this set
255.	end-of-set

Set Owner based on Current of run unit SOC

E0 = CALL (A0, "SOC, set-type")

Arguments:

set-type(input)

Currency Indicators Involved:

current of run unit(input)

current owner of set-type(output)
current member of set-type(output)

Description:

The record identified by the current record of the run unit is made the current owner of the given set-type. The first member of the set becomes the current member of that set (note that this is equivalent to an implied call to FFM).

If the set is empty, the currency indicator of the member of the given set-type becomes null, and E0 is set to 255. This routine executes an implicit FFM command; that is, the specified record is made the owner of the set and the first member associated with the new owner is made the current member of the set-type.

Errors:

01.	data base not open
02.	invalid set-type
05.	invalid owner type for this set-type
16.	no current of run unit
90.	no such DMS routine
99.	catastrophe
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
109.	user may not write this set
255.	end-of-set

Set current Owner based on current Member SOM

E0 = CALL (A0, "SOM, set-type-1, set-type-2")

Arguments:

set-type-1(input)
set-type-2(input)

Currency Indicators Involved:

current owner of set-type-1(output)
current member of set-type-1(output)
current of run unit(output)

current owner of set-type-2(input)
current member of set-type-2(input)

Description:

The record identified by the current member of the second set-type is made the current owner of the first set-type. The first member of the set becomes the current member of that set (note that this is equivalent to an implied call to FFM).

If the set is empty, the currency indicator of the member of the given set-type becomes null, and E0 is set equal to 255. This routine executes an implicit FFM command; that is, the specified record is made the owner of the set and the first member associated with the new owner is made the current member of the set-type.

Errors:

01. data base not open
02. invalid set-type
05. invalid owner type for this set-type
08. no current owner of set-type
09. no current member of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

109. user may not write this set

255. end-of-set

Set current Owner based on current Owner SOO

E0 = CALL (A0, "SOO, set-type-1, set-type-2")

Arguments:

set-type-1(input)
set-type-2(input)

Currency Indicators Involved:

current owner of set-type-1(output)
current member of set-type-1(output)
current of run unit(output)

current owner of set-type-2(input)
current member of set-type-2(input)

Description:

The record identified by the current owner of the second set-type is made the current owner of the first set-type. The first member of the set becomes the current member of that set (note that this is equivalent to an implied call to FFM). The first member of the set becomes the current member of that set (note that this is equivalent to an implied call to FFM).

If the set is empty, the currency indicator of the member of the given set-type becomes null, and E0 is set equal to 255. This routine executes an implicit FFM command; that is, the spde thecified record is made owner of the set and the first member associated with the new owner is made the current member of the set-type.

Errors:

01. data base not open
02. invalid set-type
05. invalid owner type for this set-type
08. no current owner of set-type
09. no current member of set-type
90. no such DMS routine
99. catastrophe
104. disk in wrong drive
105. disk read or write error
107. file not present
108. user may not read this set

109. user may not write this set

255. end-of-set

Set current Owner based on current Record SOR

E0 = CALL (A0, "SOR, record-type, set-type")

Arguments: record-type(input)
set-type(input)

Currency Indicators Involved:

current record of record-type(input)

current owner of set-type(output)
current member of set-type(output)
current of run unit(output)

Description:

The record identified by the current of the specified record-type is made the current owner of the given set-type.

If the set is empty, the currency indicator of the member of the given set-type becomes null, and E0 is set equal to 255. This routine executes an implicit FFM command; that is, the specified record is made the owner of the set and the first member associated with the new owner is made the current member of the set-type.

Errors:

01.	data base not open
02.	invalid set-type
03.	invalid record-type
05.	invalid owner type for this set-type
10.	no current of record-type
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set
109.	user may not write this set
255.	end-of-set

Set Record based on Current of run unit SRC

E0 = CALL (A0, "SRC")

Arguments:

Currency Indicators Involved:

current of run unit(input)
current of record-type(output)

Description:

The record identified by the current record of the run unit is set to be the current record of its record type.

Errors:

01.	data base not open
16.	no current of run unit
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present

Set current Record based on Member SRM

```
E0 = CALL (A0, "SRM, set-type")
```

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type (input)
current member of set-type (input)

current of record-type(output)
current of run unit(output)

Description:

The record identified by the current member of the specified set-type is set to be the current record of its record type.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
09.	no current member of set-type
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set

Set current Record based on Owner SRO

```
E0 = CALL (A0, "SRO, set-type")
```

Arguments:

set-type(input)

Currency Indicators Involved:

current owner of set-type(input)

current of record-type(output)
current of run unit(output)

Description:

The record identified by the current owner of the specified set-type is set to be the current record of its record type.

Errors:

01.	data base not open
02.	invalid set-type
08.	no current owner of set-type
90.	no such DMS routine
99.	catastrophe
100.	user may not read this record
104.	disk in wrong drive
105.	disk read or write error
107.	file not present
108.	user may not read this set

return data base run STATistics STAT

E0 = CALL (A0, "STAT, data-block")

Arguments:

data-block (input)

Currency Indicators Involved:

not applicable

Description:

Various operating statistics from the current execution of the MDBS.DMS system are returned in the variables of the data-block. The data-block must contain five integer variables which take on the following values:

Variable	Value
1.	# of page buffers in memory
2.	# of page references
3.	# of page faults
4.	# of disk reads
5.	# of disk writes

Errors:

01. data base not open
89. variable length inconsistency
90. no such DMS routine
94. block name not found
95. invalid data block name
96. invalid number of arguments
99. catastrophe

Notes:

1. The number of page references is the number of different times a data base page was requested by the internal routines of the MDBS.DMS system. The number of page faults is the number of times a page other than the most recently accessed page was requested.
2. All values returned are modulo 32768.

TOGGLE run optimization switch TOGGLE

E0 = CALL (A0, "TOGGLE")

Arguments:

none

Currency Indicators Involved:

not applicable

Description:

A call to the TOGGLE routine toggles the value of an internal switch in the MDBS.DMS package. When the switch is on (default), the MDNS.DMS package performs full error checking. When the switch is off, checks for certain errors which occur infrequently and have minimal impact on data base integrity, yet which require substantial amounts of processing time to perform, are bypassed. It is recommended that the switch be kept on except in well-debugged programs which work with large data bases. The switch is set to on whenever the OPEN routine is executed.

Errors:

90. no such DMS routine
99. catastrophe

IV. CONCLUDING REMARKS

This manual describes the features of MDBS in a comprehensive fashion. It may be useful to sketch out some of the extensions that are presently available. First, as the user will quickly become aware, data base requirements change over time. A logical structure that appears to be ideal during some period of time may need alteration. For example, in order to improve access efficiency new set relationships may be desirable. In some instances, new but relevant data sets appear desirable and both new record types and associated set relationships are needed. In all these cases it would be desirable not to have to restart the data base but have available dynamic restructuring capability. This capability exists with the system known as MDBS.DRS.

As the system currently exists, it is oriented towards supporting the applications programmer writing in BASIC, PASCAL, FORTRAN, COBOL, PL/I and other languages. These programs may be doing a variety of things including generating mailing lists, doing general accounting, solving linear programming problems, performing regression analyses, and forecasting to name a few. Some of the application programs could be specifically written to do data retrieval. For this class of programs it is possible to develop software which would automatically perform these tasks. A query language capability would thus obviate the user from writing application programs but would specify an English-like language for defining the desired reports. The Micro Data Base System MDBS.QRS is a high level query system /report writer having such capabilities.

Additional Software Products from Micro Data Base Systems, Inc include:

QRS (Query/Report Writer System)

QRS allows a non-programmer to extract data from any MDBS data base (a version of QRS for use with HDBS data bases is also available). QRS accepts nonprocedural, English-like queries on an ad hoc basis and automatically produces desired reports. Complex conditions on retrieval can be specified. Arithmetic expressions are also allowed. Queries can be batched. Reports can be routed to the console, a printer or a disk file. DML commands can be executed interactively. QRS is implemented in machine language.

MUS (Multi User System)

MDBS.MUS is the multiuser version of MDBS. Multiple users can share a single data base. MUS automatically saves and restores currency indicators of each executing run unit as needed. MUS also handles page lock-outs for run units that alter a data base.

VAC (Via Set, Area, Calc)

MDBS.VAC has all features of MDBS, plus features that offer the designer added control over the maping of record occurrences to storage. The designer can partition a data base into areas, each consisting of a number of logically contiguous data base pages. Record types can be assigned to specific areas. This means that all occurrences of a record type are placed in a specified area. Occurrences are mapped into an area, either on the basis of a calc

key, a via set option, or a "don't care" option. Added DML commands allow records to be directly accessed on the basis of their calc key values.

APPENDIX 1

Memory / Processing Time Trade-off

In order to assess the trade-off between the processing time required for the MDBS.DMS system to process a data base and the amount of memory available to the system, a simple experiment was performed with version 1.0 of MDBS.DMS. In this experiment, 1000 null records (i.e., records with no data items defined) were created and added to a FIFO set. The DDL for this experiment is shown in Figure 1. The optimization toggle (see DMS routine TOGGLE) was off. The data base file resided on a single Shugart-Mini disk drive in single density format.

The following observations were made:

# of pages in memory	time required (min, sec)
1	24, 18
2	16, 46
3	12, 18
4	8, 58
5	1, 26
6	0, 40
7	0, 36
8	0, 34
16	0, 29

Obviously a critical point exists where the execution time radically decreases as the number of pages available to the system increases. The extra speed realized by going from 6 pages to 7 pages is much less than that realized by going from 5 pages to 6 pages. Of course the characteristics of the data base being used and the types

of operations being performed will cause some variance in where this cutoff occurs, but we recommend that, in general, at least 8 pages be allocated if possible.

The primary cause of the differences in execution speed is due to the number of disk accesses performed by the system. Naturally, a faster disk system would result in a marked decrease in execution time.

The relationship between the number of disk accesses and the memory available to the MDBS.DMS package is in the number of page buffers that the system can allocate. The number of these buffers, then, is the number of pages which can be kept in memory in anticipation of future references. The more page buffers resident in memory, the greater is the chance that a given page will be memory resident when it is required, thus saving a disk access. The actual CPU processing time is trivial compared to the amount of time spent on I/O operations.


```
0010 FILES  DEMO            1     512
0020 DRIVE  1      100
0030 PASSWORDS
0040        PAM                              253-56-9058
0050        GARY                 255 255     665-46-9082
0060 RECORD B
0070 SET    S3         MAN   1:N
0080                                  FIFO
0090 OWNER  SYSTEM
0100 MEMBER B
0110 END
```

DDL for Experiment

Figure 1

APPENDIX 2

MDBS.DMS Command Usage

This appendix lists the commands in the MDBS.DMS system and briefly describes common uses for each command. A number has been assigned to give a relative indication of the frequency of use of each of these commands. A number such as 4 or 5 does not indicate that use of a routine should be avoided, but merely indicates the routine tends to be used rarely. The codes indicate:

1. This command is used in almost all applications.
2. This command is used extensively.
3. This command is used frequently.
4. This command is used moderately.
5. This command is used rarely.

Command Name	Usage Level	Use
ACS	1	Used whenever records are added to set occurrences
AMS	3	Alternative to ACS
CCT	5	Typically used in complex networking programs which involve a variety of record types
CLOSE	1	Always used
CMT	4	Used with sets with multiple member types
COT	4	Used with sets with multiple owner types
CR	2	Used to create null (link) records Used to defer data storage (Typically CRS is used)
CRS	1	Used to create a record and store data
DEFINE	1	Always used
DRC	3	Used to delete record occurrence
DRM	4	Typically, DRC is used
DRO	4	Typically, DRC is used
DRR	4	Typically DRC is used
EXTEND	4	Useful if a DEFINE statement will not fit on one source line
FFM	1	Basic operator for sequentially traversing a set
FFO	3	Useful with many-to-many sets
FINDM	2	General routine for locating a record with a specified data-value
FINDO	3	Many-to-many analog of FINDM
FLM	3	Useful if set is traversed "backwards"
FLO	4	Many-to-many analog of FLM
FMSK	2	Locate a record in a sorted set

Command Name	Usage Level	Use
FNM	1	Basic operator for sequentially traversing set
FNO	3	Many-to-many analog of FNM
FOSK	3	Many-to-many analog of FMSK
FPM	3	Useful if set is traversed "backwards"
FPO	4	Many-to-many analog of FPM
GETC	1	Used to retrieve all data items from a record
GETM	1	Used to retrieve all data items from a record
GETO	2	Useful when "working up" in a hierarchical data structure
GETR	2	Typically, GETM is used
GFC	1	Used to retrieve a single data item from a record
GFM	1	Used to retrieve a single data item from a record
GFO	2	Useful when "working up" in a hierarchical structure
GFR	2	Typically, GFM is used
GMC	4	Useful if only the number of members in a set is required
GOC	4	Many-to-many analog of GMC
GTC	5	Typically used in complex networking programs which involve a variety of record types
GTM	4	Used with sets with multiple member types
GTO	4	Used with sets with multiple owner types
OPEN	1	Always used
PUTC	4	Typically, PUTM is used
PUTM	2	Used to store all data items for a record
PUTO	3	Typically, PUTM is used
PUTR	3	Typically, PUTM is used

Command Name	Usage Level	Use
RMS	3	Used when altering set membership
RSM	4	Used when altering set membership
SCM	4	Allows user alteration of currency indicator
SCO	4	Allows user alteration of currency indicator
SCR	4	Allows user alteration of currency indicator
SFC	2	Used to store a single data item in a record
SFM	2	Used to store a single data item in a record
SFO	3	Typically, SFM is used
SFR	3	Typically, SFM is used
SMC	4	Typically, SMM or SMO is used
SMM	2	Useful when processing non-hierarchical structures
SMO	2	Useful when "working up" in a hierarchical data structure
SMR	3	Typically, SMM or SMO is used
SOC	4	Typically, SOR is used
SOM	2	Useful when "working down" in a hierarchical data structure
SOO	3	Useful in "forked" data structures
SOR	2	Useful in data load programs to set the current owner for AMS
SRC	4	Useful when necessary to preserve a currency indicator
SRM	5	Typically SRC is used
SRO	5	Typically SRC is used
STAT	5	Used to measure memory/disk access trade-offs
TOGGLE	4	Useful in well-debugged programs which process large quantities of data

