HEWLETT PACKARD

2100 series computers

designing IMAGE/2100 data bases

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

List of Effective Pages

Page	Effective Date
Title	October 1973
Copyright	October 1973
iii to vi	October 1973
1-1 to 1-8	October 1973
2-1 to 2-8	October 1973
3-1 to 3-2	October 1973
4-1 to 4-2	October 1973
5-1 to 5-17	October 1973
A-1 to A-3	October 1973
B-1 to B-7	October 1973
C-1 to C-2	October 1973

Printing History

First Edition . . . . . October 1973

Preface

This manual describes the Data Base Definition Language (DBDL), Data Base Definition System (DBDS), and Data Base Utility System (DBUS) of the Hewlett-Packard *Information Management System for HP 2100 Computers* (IMAGE/2100). IMAGE/2100 executes under the control of DOS-III: Disc Operating System. Users of this manual should be familiar with the following publications:

- Decimal String Arithmetic Routines (02100-90140)
- DOS-III: Disc Operating System (02100-90136)
- Query: On-Line Access to IMAGE/2100 Data Bases (02100-90142)
- Using IMAGE/2100 Data Bases (02100-90139)

Section I of this document describes the IMAGE/2100 system and the IMAGE/2100 data structure. Section II details the Data Base Definition System (DBDS) and the Data Base Definition Language (DBDL). DBDS control options appear in Section III, while Section IV describes step-by-step procedures for DBDS operation. Section V explains the Data Base Utility System, including the use of the DBBLD program, used for storing data in a data base. Appendix A provides a working example of DBDS and DBDL; Appendix B lists the error messages that may occur during DBDS operation, and Appendix C describes the error conditions that may occur during operation of the Data Base Utility System (DBUS).

Contents

FIGURES

TABLES

SECTION I
Introduction

IMAGE/2100

The Information Management System for 2100 Computers (IMAGE/2100) is a set of software subsystems designed to provide a general purpose framework for defining, accessing, modifying, and reporting on data arranged into structures known as data bases.

IMAGE/2100 Components

The IMAGE/2100 system is comprised of four separate subsystems:

- Data Base Definition System (DBDS)
- Data Base QUERY System (DBQS)
- Data Base Management System (DBMS)
- Data Base Utility System (DBUS)

DBDS. DBDS is the data base design subsystem used to create the structure of a data base. The user defines the data base in the Data Base Definition Language (DBDL), which is then processed by DBDS to create a data base framework on a computer file.

DBQS. DBQS (QUERY) is a program used to aid in on-line data storage, update, and retrieval from a terminal device. By entering English-like commands on the keyboard terminal device, the user accesses the data base without any programming effort.

DBMS. The Data Base Management Subsystem is a set of subroutines called by either DOS-III Fortran or HP 2100 Assembly Language programs. By using the subroutines to open, read, write, and delete data from the data base, the user can programmatically manipulate data in the data base without concern for the data base structure.

DBUS. Copies existing data base structures and actual data contained in the data base onto magnetic tape and then restores the data base from magnetic tape back to the DOS-III files. A module of DBUS called DBBLD, stores data into a data base from cards, magnetic tape, paper tape, or from a DOS-III file. This is useful for initially storing large amounts of data into a newly created data base, or for updating a data base with new data entries.

In addition to the subsystems described above, users of IMAGE/2100 require the Extended File Management Package (EFMP) and the Decimal String Arithmetic Routines. IMAGE/2100 uses EFMP for greater file handling capability and stores actual data base values in EFMP files. Calculations performed on data base values (especially by the Data Base QUERY subsystem) require the Decimal String Arithmetic Routines.

Environment

IMAGE/2100 operates under the control of the Disc Operation System for 2100 Computers (DOS-III), which allows single-user access to the system. The DOS-III hardware and software configuration is flexible, but requires the following minimum hardware for IMAGE/2100 operation:

- HP 2100 computer with a minimum of 16K core memory.
- Single terminal (which is the system terminal).
- Single-platter Moving-head disc.

The system can also include a line printer and a magnetic tape unit for use by DBUS for producing back-up copies of the data base.

Using IMAGE/2100

Operation of IMAGE/2100 occurs in several stages:

1. First, the user defines a data base structure using the Data Base Definition Language. This definition of the data base is called a *schema.*
2. Next, the user defines two files using the DOS-III :STORE,B directive. One file is a regular DOS-III file and is used for storing the data base root file, while the other file is an EFMP file (named PN*xxx*) which contains the actual data base values.

3. After the files are created, the user invokes the Data Base Definition System, which processes the schema (written on cards, paper tape, or a DOS-III disc file). DBDS creates a data base structure called the *root file* which is stored in one of the files created prior to DBDS execution.

4. After the root file is created, the user is ready to access and use his data base in one of three ways:

   a. Use the DBBLD program (a module of DBUS) to store data entries into the newly-created data base.

   b. Invoke the DB QUERY subsystem through a DOS-III command and enter values into the data base through the system keyboard terminal device.

   c. Execute a user Fortran or Assembly Language program which uses the DBMS subroutines to access the data base and transfer data into the data base.

5. After the user has entered data into his data base through one of the methods outlined in step 4, he can access and modify the data base using one of the same three methods:

   a. Use the DBBLD program to append new data entries to the data entries already stored in the data base.

   b. Use the QUERY subsystem to read, replace, report, or delete entries from the data base.

   c. Execute a user Fortran or Assembly language program which uses the DBMS subroutines to access the data base and read, replace, report, or delete entries from the data base.

6. If the user is the data base manager, and the DOS-III system includes a magnetic tape unit, he can store the data base on magnetic tape for backup purposes in one of two ways:

   a. Store the actual data base entries on magnetic tape without the data base structure. When the data base is restored to the DOS-III files, the user has the option of redefining the data base structure and loading the data entries into that structure.

   b. Store the data base entries and the data base structure (root file) on magnetic tape (sector-by-sector), and restore the data base entries and structure exactly as they were copied.

IMAGE/2100 DATA BASE ORGANIZATION

Data Item

The smallest unit of data accessible to the user in a data base is the *data item.* The *data item* consists of a name (known as the *attribute*), and a value. The name of the data item, type of internal representation and the security level for reading and writing the item are defined in the data schema. Data items can consist of an integer value one 16-bit computer word long, or a real value two 16-bit computer words long, or as a character string of variable length up to 63 words (126 characters) long.

Data Entry

A *data entry* consists of one or more *data items*, each item having a unique name. The data entry contains only the values of the data items, stored in the order defined in the *data schema*. The attributes of the data items are stored in the root file while the values of the data entries are stored in the data base file.

Data entries exist internally as a logical record consisting of the item values (data record) and system overhead made up of pointers and header words (media record). (The pointers and header words appearing in the media record are described below.) The size in computer words of a data entry varies from data set to data set depending upon the number and size of data items and the number of words in the media record. Figure 1-1 shows a representation of a data entry. The entry includes a media record and the values of the data items. The names of the data items are shown for clarity, but they are not stored in the data entry.

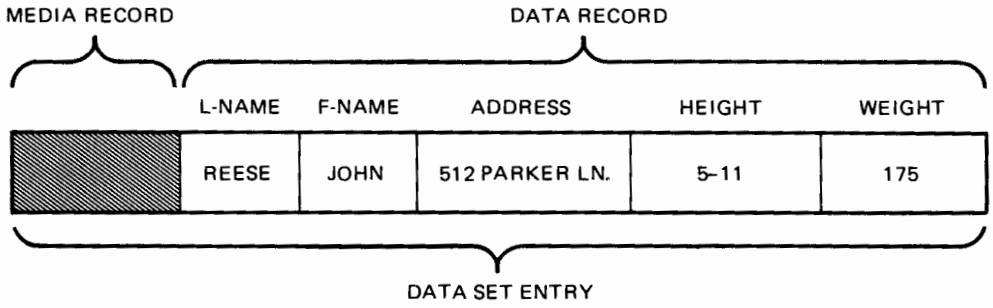


Figure 1-1. Data Entry Representation

Data Set

A *data set* is a named collection of data entries. Figure 1-2 shows a representation of a data set.

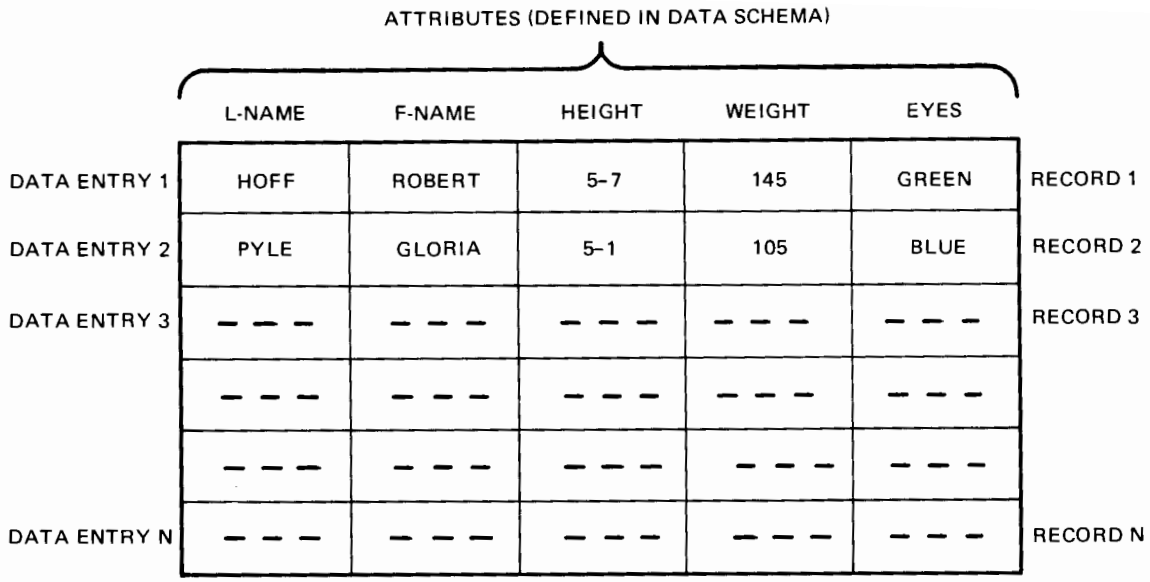


Figure 1-2. Data Set

Two types of data sets can be defined under IMAGE/2100 — *master data sets* and *detail data sets.*

During creation of the data schema, a detail data set is defined with none, one or more *key items.* A *key item* is a data item name specified by the user to link the detail data set with a specific master data set through a *key item* defined for the master data set. A detail data set can be defined with a maximum of five different key items, each key item referencing a different master data set. Figure 1-3, for example, shows a detail data set with two key items (named EYES and HAIR) defined. The EYES key item links the detail data set with the "Eyes" master data set and the HAIR key item links the detail data set with the "Hair" master data set. Data entries are loaded into detail data sets in a sequential manner. The first detail data entry loaded is assigned record address 1 and subsequent entries are assigned record address 2, 3 and so on.

If an entry is deleted from a detail data set, IMAGE/2100 fills the location of the deleted entry with a new data entry (when a new data entry is entered by the user for that detail data set). This method of allocation is known as *serial allocation.*

Detail data sets are constructed so that a pointer points from a data entry with one key item value to the next data entry with the same key item value, and from a data entry with one key item value to the previous data entry with the same key item value. The first occurrence of a data entry with a specific key item value has no backward pointer, while the last occurrence of a data entry with a specific key item value has no forward pointer. Like key item values and their forward and backward pointers form a *data chain.* Data chains divide data sets into subsets based on the value of the key items. In Figure 1-3, for example, the key item called EYES in the detail data set contains four data chains based on the four different key item values GREEN, BLUE, BROWN, and GREY. The key item called HAIR contains four data chains based on the key item values BROWN, BLOND, GREY, and BLACK.

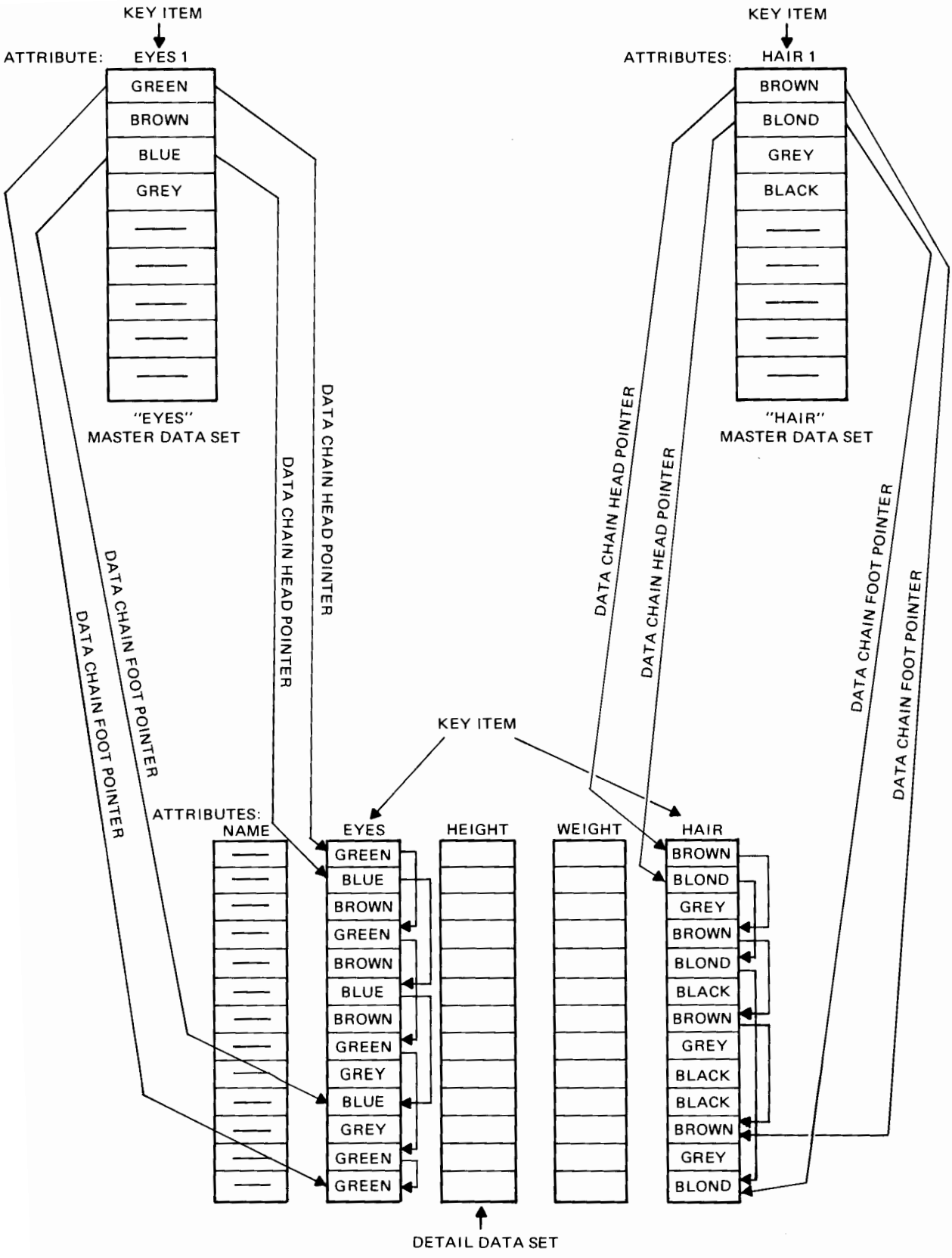


Figure 1-3. Master – Detail Data Set Relationships

Master data sets must be defined with exactly one key item. This key item can act as a link to as many as five different detail data sets. Unlike detail data sets, master data set entries contain unique key item values. Figure 1-3, for example, shows several occurrences of the value GREEN in the key item EYES for the detail data set, while the "Eyes" master data set contains the key item value GREEN only once. Data entries are loaded into master data sets in a random fashion within the alloted data space. IMAGE/2100 uses a key transformation formula to assign record addresses to new data entries based on the value of the key item of the data entry. (Conversely, IMAGE/2100 uses the key transformation function to access data entries with a specific key item value.)

For every key item value occurring one or more times in a detail data set, the same key item value occurs once and only once in the master data set linked to the detail data set. Along with the key item value, the master data set also contains two pointers, one pointing to the head of the data chain (the first entry of the chain) and the other pointing to the foot of the chain (the last data entry of the chain). All the key item values and their related pointers to the foot and head of the data chains form a *data path* from the master to the appropriate detail data set. Up to five *data paths* can be defined for the same master data set. These *data paths* link the master to five separate detail data sets. For example, Figure 1-3 shows a data path between the "Eyes" master and the detail data set and between the "Hair" master and the detail data set.

Using Master Data Sets

Use of master data sets lessens the retrieval time necessary to obtain desired information from a detail data set during operation of the QUERY subsystem. Master data sets do this by partitioning the data set into subsets based on the value of key data items. By using the pointers in the master data set to the head of the data chain for a specific key item value, QUERY accesses only those data entries in the detail data set which contain the desired key item value. Thus, QUERY avoids time-consuming entry-by-entry searches, through the entire data base. Using Figure 1-3 as an example, if the user wishes to retrieve all data entries with the value of data item EYES equal to GREEN, the QUERY subsystem would perform the following steps:

- Using the data schema defined for the data base, QUERY determines which master data set is linked to the detail data set through the key item named EYES.

- Having found the appropriate master data set, QUERY uses the key transformation function to locate the data entry containing the key data item value equal to GREEN in the master data set.

- QUERY then reads the pointer associated with the key item value GREEN, which points to the head of the data chain in the detail data set which also contains key item values equal to GREEN. QUERY accesses the first data entry (Head of chain) indicated by the pointer.

- QUERY accesses the next data entry with key item EYES equal to GREEN by reading the forward pointer associated with the current data entry.

- QUERY continues to access all data entries in the data chain until the last data entry in the data chain has been reached. The last data entry in the chain contains no forward pointer to another data entry.

Although using master data sets decreases retrieval time, an increase in storage space within the data base file results. The number of pointers and linkages stored in the media record of a data entry increases for each data path defined between a detail data set and a master data set. For a master data set, the number of computer words in the media record is 3 + (3 * *path count*); for a detail data set, the number of words in the media record is 1 + (2 * *path count*) where *path count* is the number of paths defined for the data set.

For example, the detail data set in Figure 1 - 3 would have a media record length of 1 + (2 * 2) = 5 while each one of the two master data sets would have a media record length of 3 + (3 * 1) = 6. This system overhead can significantly increase the amount of storage space needed to store the data base.

SECTION II
Data Base Definition System

The Data Base Definition System (DBDS) processes a description of the user data base (called a schema) written in the Data Base Definition Language (DBDL) and produces an internal system description of the data base (called a *root file*) used by the IMAGE/2100 system. This *root file* describes the relationship between data items, the level of security associated with each item and the relationships between master and detail data sets in the data base. The root file exists as a DOS-III file (non-EFMP) which must be defined prior to DBDS execution. Using DBDS consists of four main steps as shown in Figure 2-1:

1. Design of the data base by the data base manager.

2. Describing the design in the Data Base Definition Language (creating a *schema*),

3. Processing the data *schema* by DBDS,

4. Creation of the root file by DBDS.

DBDS operation is outlined in Section III.

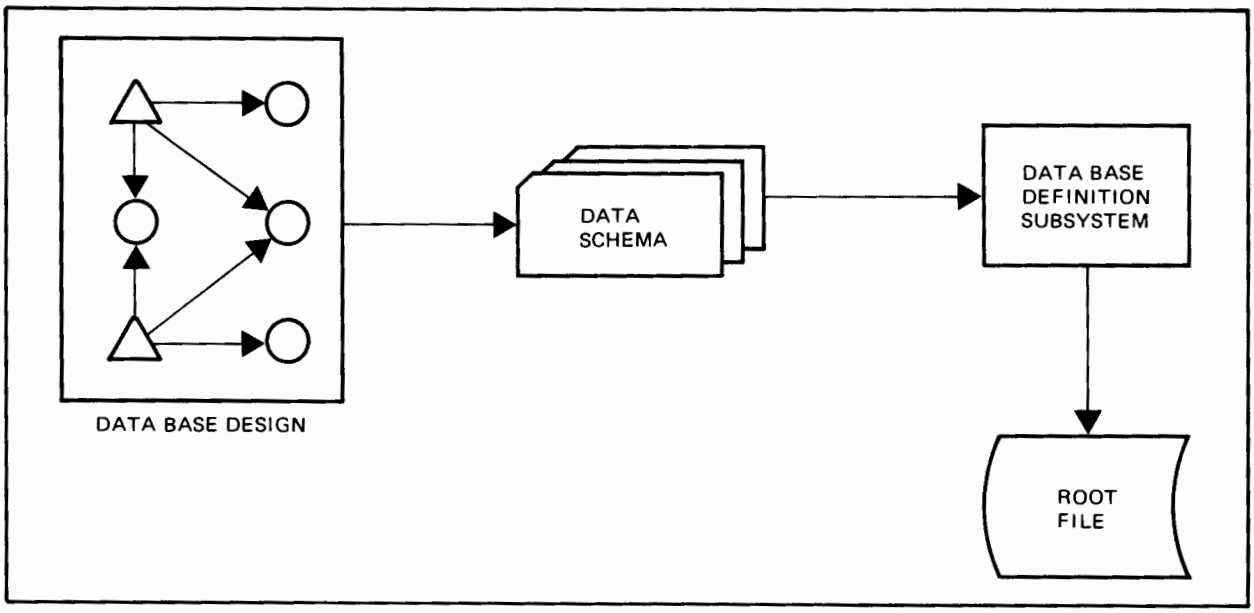


Figure 2-1. Data Base Design Process

DATA BASE DEFINITION LANGUAGE

The Data Base Definition Language (DBDL) is the language used to define a data base schema to be processed by the DBDS. The data schema exists in the DOS-III system as an ASCII file, either on cards, tape, or as a catalogued disc file. The data schema should be written in 80-column fields, although DBDS processes only columns 1 through 72; columns 73-80 can be used for sequence numbers. Because DBDL is a free-format language, the user can insert blanks anywhere in the data schema to improve its appearance except within symbolic names, reserved words or other syntactical entities.

The following conventions appear in the DBDL descriptions below:

1. Information in upper case, italicized letters appears in a schema exactly as shown. For example,

   SETS:

2. Lower-case italicized words stand for a class of entities. For example,

   path count

   is defined as an integer between 1 and 5.

3. Information in square brackets ([ ]) is optional depending upon the user's needs. For example,

   item name [(path count)]

   means either:

   item name (path count)

   or

   item name

4. Information in braches ({ }) indicates that one member of a group must be chosen. For example:

   NAME: setname, $\left\{ \begin{matrix} DETAIL \\ D \end{matrix} \right\}$;

   means either:

   NAME: setname, DETAIL;

   or

   NAME: setname, D;

Comments

The user can document his data schema through the use of comments, which may appear anywhere in the data schema. Comments are listed with the data schema but are otherwise ignored by DBDS. A comment may not be imbedded within another comment. Comments take the form:

<<comment>>

where *comment* is an alphanumeric character string.

Examples of comments are:

```
<<THIS IS A COMMENT>>

<<COMMENTS CAN APPEAR ANYWHERE WITHIN A SCHEMA>>

<<COMMENTS ARE IGNORED BY DBDS>>
```

Data Base Schema Framework

The actual content of a data base schema varies for each data base depending upon the needs of the data base designer. The structure of the data base schema does not vary. The data base schema structure is:

EFMP PACK ID packnumber; security code;
BEGIN DATA BASE data base name;
LEVELS: [level part]
ITEMS: item part
SETS: set part
END.

packnumber — is an alphanumeric string of five characters beginning with the letters PN followed by a three digit integer from 001 through 999.

security code — is an integer between 1 and 32,767 inclusive. This number is chosen by the data base designer as a password. Any user attempting to access the data base must enter the *security code* when Data Base QUERY prompts for it.

data base name — is a string of one to five printable characters except the at-sign (@) or the semi-colon (;). *data base name* must be the same as the name chosen by the user for the DOS-III file which will hold the root file created by DBDS. The DOS-III file must be created prior to DBDS execution (using a :STORE,B . . . directive.).

Level Part Definition

The *level part* of the data schema is used to supply privacy levels and code words to access those levels for specific data items specified in the *item part* of the data schema. If the *level part* is omitted from the data schema, then no privacy levels exist for specific data items in the data base. Each privacy level is defined using a level number and a level code word. The level number is an integer from 1 to 15 and must appear in the *level part* in ascending order, while the level code word consists of from 1 to 6 ASCII characters (left-justified in the field),[1] or may be omitted altogether. Privacy levels are separated by a semi-colon; level numbers and level code words are separated from each other by one or more blanks. Any *level part* that is not empty (i.e., that specifies one or more privacy levels) must define a privacy level with a level number equal to 15, and a non-empty level word.

The form of the *level part* is:

level number [level word];
level number [level word];
.
.
.
level number [level word];

Below is an example of the level part of a data schema. Notice that the last privacy level number is 15 as required by the DBDS, and that the level word for privacy level 4 is empty (omitted).

```
LEVELS:

    1       WRITER ;

    4              ;

    8       MANAGE ;

    15      PRESDT ;
```

Item Part Definition

The *item part* of the data schema defines the names, size and method of representation in memory, and privacy levels (one for reading and one for writing) of all the data items in a data base, regardless of the data set to which the items belong. Up to 100 data items can be defined in a data schema *item part*. The definition sequence for each data item takes the following form:

Item name , spec (read level, write level) ;

[1] Any ASCII character may be used except null, blank, line feed, return, X-off, alt-mode, escape, left or right parenthesis, less than (<), greater than (>), comma (,), semi-colon (;), or rubout.

where:

item name — is the name of the data item consisting from one to six characters starting with a letter of the alphabet. The remaining five characters (if any) consist of letters (A-Z), digits (0-9), or any of the following ASCII characters: (+, -, *, /, ?, '. #, %, &, @). Each *item name* in the *item part* must be unique.

spec — defines the amount of space in memory that the data item is to occupy, (i.e., the data *item type*). Three data types are allowed as the *spec* for data items:

- R2 denotes a REAL (floating point) number occupying two words in memory.
- I1 denotes an INTEGER number occupying one word in memory.
- U *integer* denotes an ASCII character string containing no lower case alphabetics. The *integer* following U (with no intervening blanks) denotes the number of characters or bytes (a byte is a half-word) in the string, and must be an even number not exceeding 126.

(read level, write level) — is an ordered pair of integers specifying the privacy level needed to read the data item and the privacy level needed to modify the data item. (If read and write levels are omitted, DBDS assigns a read level of zero and a write level of 15 to the data item.) During creation of the *level part* of the data base, the data base designer specifies a series of privacy level numbers and privacy level code words. The *read level* and *write level* are integers corresponding to privacy level numbers from 1 to 15. These numbers need not be mentioned in the level part of the schema. When a user first executes the Data Base QUERY System, he must enter a privacy level code word which corresponds to the highest level number the user is allowed to access. The user cannot read or modify a data item unless his privacy level number is equal to or greater than the *read level* or *write level* number specified for the data item in the *item part* of the data schema. The *read level* must be less than or equal to the *write level* which must be between 1 and 15, inclusive. A data item used as the key item in a data set must have a write level of 15 (if the read and write levels are listed with the data item).

An example of an *item part* of a schema is:

```
ITEMS:
   NAME,     U20(6,6)   ;  <<NAME IS TYPE CHARACTER>>
   HEIGHT,   I1         ;  <<HEIGHT HAS READ - 0, WRITE - 15>>
   WEIGHT,   I1(6,6)    ;  <<WEIGHT IS TYPE INTEGER>>
   EYES,     U8(6,6)    ;  <<EYES IS 8 CHARACTERS LONG>>
   OUTPUT,   R2(1,4)    ;  <<OUTPUT IS A REAL NUMBER>>
```

Set Part Definition

The *set part* of the data schema names the various data sets within the data base, indicates which items listed in the *item part* of the schema belong to which set and links the master data sets to the detail data sets by describing which of the data items are *key items.* The data schema can define at most, 20 data sets. No data entry (DATA RECORD plus MEDIA RECORD) can contain more than 256 words.

Master Data Sets

The form of the *set part* for definition of master data sets is:

```
NAME:       setname, type;
ENTRY:
            item name  [(path count)] ,
            item name  [(path count)] ,
            .
            .
            .
            item name  [(path count)] ;
CAPACITY:   capacity count ;
```

where:

setname is the name of the set consisting of from one to six characters starting with a letter of the alphabet. The remaining five characters (if any) consist of letters (A-Z), digits (0-9), or any of the following ASCII characters: (+, -, *, /, ?, ', #, %, &, @). The combination of the first five letters must be unique. The sixth letter is ignored by DBDS.

type is either MANUAL (or M) or AUTOMATIC (or A). A data entry with a key item value must be inserted by the user into a MANUAL master data set prior to inserting (for the first time) a data entry with the same key item value into an appropriately linked detail data set. For an AUTOMATIC master data set, IMAGE/2100 automatically inserts a master data entry whenever a detail data entry is inserted with a key item value unique to the detail data set and not already present in the master.

item name is the name of a data item belonging to the master data set. The name must have previously appeared in the *item part* of the data schema. *item names* in the list under ENTRY: are stored by IMAGE/2100 in the order they are listed and must appear once and only once within the *set part* of the schema. One and exactly one *item name* must be followed by a *path count* to identify the item as a key item link to a detail data set. The data item must be the same type and length as the key data item in the detail set linked to the master set. An AUTOMATIC master set must contain one and only one data item (which must be the key item and contain a non-zero *path count*).

path count is an integer from 0 to 5 (inclusive) which indicates that the data item is a key item. The number itself indicates how many paths exist from the master data set to detail data sets through the key item link. Thus, if the master is linked to two separate detail data sets, the *path count* is 2. A *path count* of 0 (allowed only for a MANUAL master) is used when a manual master set is used for information only and not for linking to a detail data set.

capacity count is an integer between 1 and 32,767 indicating the number of records allocated to the data set.


```
THE FOLLOWING ARE TWO EXAMPLES OF MASTER DATA SETS AS THEY MIGHT
APPEAR IN THE SET PART OF THE SCHEMA:

<<EXAMPLE ONE>>

NAME:      COLOR,A; <<"A" STANDS FOR AUTOMATIC>>

ENTRY:     EYES1(1); <<AUTOMATIC MASTERS CAN CONTAIN ONLY ONE ITEM>>
                     <<WHICH MUST BE THE KEY ITEM IN THIS EXAMPLE.>>
                     <<THE ITEM IS LINKED TO ONLY ONE DETAIL SET,>>
                     <<SO THE PATH COUNT IS "ONE".>>

CAPACITY: 10;       <<<THIS MASTER MAY CONTAIN UP TO 10 ENTRIES>>


<<EXAMPLE TWO>>

NAME: SCHOOL,MANUAL;

ENTRY:
      NAME,      <<MANUAL MASTERS CAN CONTAIN ONLY ONE KEY  ITEM, BUT>>
      NUMBER,    <<OTHER INFORMATION CAN BE INCLUDED AS WELL.>>
      CITY(2),   <<CITY IS THE KEY ITEM LINKED TO TWO DETAIL SETS.>>
      SIZE;

CAPACITY:  250;
```

Detail Data Sets

Detail sets are defined in the data schema in almost the same form as master data sets as follows:

NAME: *setname*, $\left\{ \begin{matrix} DETAIL \\ D \end{matrix} \right\}$;

ENTRY:

item name [(link)] ,
item name [(link)] ,
.
.
.
item name [(link)] ,

CAPACITY: *capacity count* ;

where:

setname is the name of the set and follows the rules for master data set names. The word DETAIL (or the letter D) follows the *setname* after a comma to indicate that the set is a detail data set instead of a master data set.

item name is the name of a data item belonging to the detail data set. The name must have previously appeared in the item part of the data schema and must appear once and only once in the schema *set part*. *item names* in the list under ENTRY: are stored by IMAGE/2100 in the order they are listed. A data item used as a key item must be of the same length and type as the corresponding key item in the master data set linked to the detail.

link is the *setname* of a master data set. When a *link* follows the item name in a detail data set, this indicates that the data item is a key item. The *link* indicates which master data set is linked to the detail set. Up to five such key items can be defined for each detail data set. If none of the *item names* in the list have a *link* following, the detail data set is not linked to any master set. All *links* following *item names* must be those of previously defined masters. For this reason, it is useful to define master data sets before detail data sets in the *set part* of the data schema.

capacity count is an integer between 1 and 32,767 indicating the number of records allocated to the data set.

An example of a detail data set defined in the data schema *set part* is:

```
NAME: WORKER, DETAIL;
ENTRY:
      NAME,
      AGE,
      SKILL(JOBTYP), <<THIS IS A KEY ITEM LINKING TO MASTER "JOBTYP">>
      SALARY(MONEY); <<THIS IS A KEY ITEM LINKING TO MASTER "MONEY">>
CAPACITY:  1000;      <<THIS DETAIL CAN CONTAIN UP TO 1000 ENTRIES.>>
```

SECTION III
Schema Control Options

The schema CONTROL command is an optional command used by the data base designer to specify control options during processing of the data base schema by the DBDS. The command is entered as the first record of the schema and its form is:

$CONTROL [parameter,parameter,...parameter] ;

where *parameter* is any of the following:

LIST	List each source record on the list device.
NOLIST	Suppresses the LIST option. When an error occurs, the offending source record is printed along with an error message.
ERRORS=nnn	Sets the maximum numbers of errors allowed to nnn (0<nnn<999). If this number is exceeded during processing, the schema processor terminates, after printing the message: MAX ERRORS — SCHEMA PROCESSING TERMINATED
ROOT	Informs the schema processor to create a root file if no errors are detected in the schema.
NOROOT	Prevents the schema processor from creating a root file. The schema is merely checked for errors.
TABLE	The schema processor prints a table of summary information (if no errors were detected) about the data sets after the data schema has been scanned by DBDS. (See Appendix A for sample printout.) The information printed is as follows:
DATA SET NAME	The name of the data set.
TYPE	A for automatic, D for detail, M for manual.

FLD CNT	Field count. The number of data items in a data set entry.
PATH CNT	Path count. The number of paths defined for the data set.
ENTR LGTH	The combined length (in computer words) of all the data items in the data entry.
MED REC	Media Record Length. The length of the media record for each entry in the data set, calculated by 3 + (3 x *path count*) for master data sets and 1 + (2 x *path count*) for detail data sets.
CAPAC CT	Capacity count. The number of entries allowed for each data set as defined in the data schema.

NOTABLE Suppresses printing of the summary table.

Some examples of $CONTROL commands are:

```
$CONTROL NOROOT,TABLE;
$CONTROL ERRORS=2;
```

Unless otherwise specified in a $CONTROL command, the schema processor operates with the following default conditions:

```
LIST
ERRORS=100
ROOT
NOTABLE
```

SECTION IV
DBDS Operation

To operate the Data Base Definition System and process a data base schema, perform the following steps:

1. Create an ASCII file containing the data base schema defined in the Data Base Definition Language. The file can be on cards, paper tape, or stored on the disc as a source file.

2. If DBDS is to create a root file (the ROOT control option is active), prepare the root file space by entering the DOS-III directive:

   :STORE,B, data base,n

   where:

   data base is the same as the *name* in the data base schema statement BEGIN DATA BASE *name*; *n* is the number of sectors in the file. The root file will be at most 10 sectors long.

3. If the schema file exists on paper tape or cards, place the file in the input device and ready that device.

4. Start DBDS execution by entering the DOS-III directive:

   :PROG,DBDS, logical unit

   where:

   logical unit is the DOS-III logical unit number of the input device containing the schema file.

   a. If the user does not enter the logical unit number when entering the :PROG directive, DBDS responds in the system console with:

      PARAMETER FOR LOGICAL UNIT NOT SPECIFIED.

      Re-enter the :PROG directive including the logical unit number.

b. If the *logical unit* specified in the :PROG directive is 2 (disc), DBDS responds on the system console with:

ENTER FILE NAME

Respond by typing the DOS-III file name containing the schema on the disc. If the :PROG directive is entered in batch mode, the record following the :PROG directive (on the batch device) must contain the file name. If the file entered by the user contains a schema file, DBDS continues execution with step 5. If the file name entered does not contain a schema file, DBDS responds on the system console with:

SCHEMA FILE DOES NOT EXIST UNDER SPECIFIED NAME *name*

Respond by returning to the beginning of step 4.

5. DBDS reads and processes the data schema according to the control options specified in the $CONTROL record (see Section III). If no $CONTROL record exists, DBDS operates under the default conditions specified in Section III.

6. If errors occur during schema processing (see Appendix B), DBDS prints error messages on the list device. Correct the schema and return to step 3 of the operating instructions.

7. When DBDS terminates execution, it types the message:

```
DBDS    :STOP    0000
```

SECTION V
Data Base Utility System

The Data Base Utility System (DBUS) consists of five routines primarily used by the base manager for general maintenance of data bases. This includes loading large amounts of data into a newly created data base, copying data from a data base onto magnetic tape with the option of restructuring the data base, or simply copying the data base structure and data to magnetic tape, sector by sector. The five DBUS routines and their uses are:

- DBBLD

  Loads actual data entries into a data base. DBBLD is useful for initially storing large amounts of data into a data base, or adding data entries to data already stored in a data base.

- DBULD

  Copies data entries from a data base onto a magnetic tape file. Unloading the data base using this routine allows the user to reload the data base into a different data base structure.

- DBLOD

  Loads data entries into a DOS-III file from a magnetic tape file created by the DBULD routine. DBLOD users can restore the data to the same data base structure or create a new data base structure using a new data base schema.

- DBSTR

  Copies the data base root file and data base entries onto magnetic tape, sector by sector, which also loads any pointers associated with the data base.

- DBRST

  Restores a root file and associated data base entries into DOS-III files from a magnetic tape file created by the DBSTR routine. Unlike DBULD and DBLOD, DBRST only restores the contents of the magnetic tape. No modification of the data base structure is allowed.

DBBLD

DBBLD loads actual data item values into a data base structure from cards, magnetic tape, paper tape, or a DOS-III file. Besides storing data, DBBLD confirms that the data is presented in the proper format and lists the data on a list device. DBBLD is useful for initially storing large amounts of data into a newly-created data base or adding data entries to data already stored in a data base. The data can be prepared away from the computer (on punch cards, for example) and then loaded conveniently on DBBLD.

DBBLD Operation

DBBLD is invoked through the DOS-III directive:

:PROG,DBBLD,p_1,p_2,p_3,p_4,p_5

where:

p_1 = the logical unit number of the device containing the data base file to be read and stored by DBBLD.

p_2 = the logical unit of the list device.

p_3 = is the mode parameter:

1 indicates that DBBLD is to check the data base file for correct form. No data is actually stored in the data base.

3 indicates that the entries in the data base file are to be added to the entries already existing in the data base.

5 indicates that the data base contains no data and the data entries in the data base file are used to initially create the data base.

p_4 = 0 if the user wishes a listing of the data as it is read by DBBLD.

1 if the listing of the data is to be suppressed.

p_5 = The desired number of columns of a data record to be read by DBBLD. The number must be greater than zero and less than or equal to 510. DBBLD reads an entire physical record (such as 80 column cards) but processes only the number of columns specified by p_5. The remaining columns in the record (if any) are ignored by DBBLD.

If p_2 = 2 (indicating that the data file exists on the disc as a DOS-III file), DBBLD prompts the user for the name of the file by typing:

ENTER FILE NAME

Type the name of the file on the keyboard device (if operating in keyboard mode) or insert the name of the file on the record immediately following the record containing the :PROG,DBBLD DOS-III command (if operating in batch mode).

After the DOS-III directive is issued, DBBLD reads the data file. If the user requested a listing, the data is listed as it appears in the file. If an error occurs, a message is printed out under the offending record in the listing. If no listing was requested and an error occurs, DBBLD prints the error message following the offending record. See Appendix C, "*DBUS Error Messages.*"

Prior to invoking DBBLD (with p_3 = 3 or 5), create the data base file which will contain the actual data by entering the DOS-M directive:

:STORE,B,PNxxx,n

where:

PNxxx is the name of the EFMP file defined in the data base schema containing the actual data for the data base. (This is not the same file as the *root file.*)

n is the length of the data base file in sectors. The length is reported by DBDS after processing the data base schema by printing the message:

DATA BASE LENGTH: *xxx* SECTORS

If a fatal error occurs while DBBLD is loading data into the data base file (p_3 = 5), purge the data base file using a DOS-III :PU directive and re-open the file using the DOS-III :STORE,B directive.

If during DBBLD operation the $END record is missing from the data base file (when the file is on cards or tape), the DOS-III error message

DEVICE *xxx* DOWN

occurs. If the data file is on cards, place a $END card in the reader, type the DOS-III directive :UP,*xxx* (*xxx* is the logical unit number of the device), then type :GO. DBBLD will read the $END record, store the data in the data base file and halt. If the data file is on paper tape, repunch the tape including the $END record and rerun DBBLD.

If the $END record is missing from the data base file on the disc, error message 209 results. Consult Appendix C, "DBUS Error Messages" for the appropriate action.

To run DBBLD using a data file on magnetic tape, make sure that the magnetic tape file contains an end-of-file mark at the end of the file.

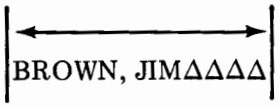
When entering large amounts of data into a data base, it is strongly suggested that the user run a test containing values for only one entry for each data set in the data base. By setting parameter p_3 = 1 in the DOS-III directive :PROG,DBBLD, the user can confirm that the data fields have been correctly entered in the DBBLD input file prior to loading data into the data base.

Data File Format

DDBLD reads a data file consisting of physical records. Columns 1 through P_5 are used for data, while remaining columns are ignored by DBBLD and can be used for sequencing information. Individual data item values are grouped together according to the data set to which they belong. Data for a specific data set is identified by a $SET: *setname* record preceding the data records. (*setname* is the name of the data set.) Data item values appear in the data file in the order they are listed in the schema set definition.

Columns 1 through P_5 of each data file record are divided into fields whose lengths depend upon the data item type. Type I1 (integer) data item values are right-justified in a five-column field, while type R2 (real) data item values are right-justified in a ten-column field (with no decimal point). Type U (character) data item values appear in fields exactly as long as the defined item length. For example, a value for an item defined as U30 appears in a field 30 columns long. Character values need not be justified in their fields, but leading and trailing blanks in the field are significant and count as part of the data item value. For example:

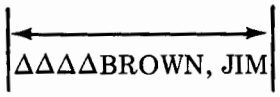
Character - Value Field



Δ = ASCII "blank"

is not the same as:

Character-Value Field



All the fields pertaining to a data entry in the set (remember that a data entry is a collection of data items) are joined together without intervening spaces in the data file records. The field for the first data item in a data entry must start in column 1 of a DBBLD data file record. The fields continue through column P_5 of the record unless a field extends beyond column P_5. In that case, the field must start in column 1 of the next record. If a single field is more than P_5 columns long, the field must start in column 1 of a record, extend through column P_5, and continue with column 1 of the next record. For example, a value for a data item named COMENT defined as U90, would appear in the DBBLD data file as follows (in this case, the file is on 80-column cards with P_5 = 72):

	1 ... 72
RECORD 1	THE DESIGN MEETING FOR THIS PROJECT IS SCHEDULED FOR APRIL 29, 1973.WORK
2	BEGINS LATER.

When entering values into the data base, not every data item need receive a value. To omit entering a value into the data base, for a specific data item, leave the field for that data item blank.

Negative Numbers

Negative numbers entered into the data base as I or R type data item values must appear in the data base file in a special way. Rather than insert a minus sign (–) in front of a number to show that the number is negative, the user must alter the last digit of the number according to Table 5-1 below. For example, according to Table 5-1, the number –1004 appears as 100M in the data base file.

Table 5-1. Representing Negative Numbers in DBBLD Data Base Files

If the number is negative and the last digit is a:	The last digit of the number should be represented as:
0	- (minus sign)
1	J
2	K
3	L
4	M
5	N
6	O
7	P
8	Q
9	R

Data File Initial Record

The entire data file is identified by an initial record signifying the name of the data base, the security code for the data base and if defined in the data base schema, the level 15 code word. This record starts in column 1 and assumes the following form:

data base name, security code, level word;

or

data base name,security code;

For example, the record might look like:

CABCO,1234,ADMIN;

The last record of the DBBLD data file must be $END (starting in column 1.).

Figure 5-1 shows part of a data schema with items and set definition. Figure 5-2 shows a DBBLD data file containing data for the data base (on 80-column cards with P_5 = 72).

```
EFMP PACK ID PN500;1000;
BEGIN
DATA BASE CABCO;
LEVELS:
ITEMS:
            NAME,              U30;
            ADDRES,            U30;
            PHONE,             U8;
            OCCUP,             U15;
            SALARY,            I1;
            SALES,             R2;
SETS:
NAME:       EMPLOY,DETAIL;
ENTRY:
            NAME,
            ADDRES,
            PHONE,
            OCCUP,
            SALARY,
            SALES;
CAPACITY:   1000;
END.
```

Figure 5-1. Data Base Schema

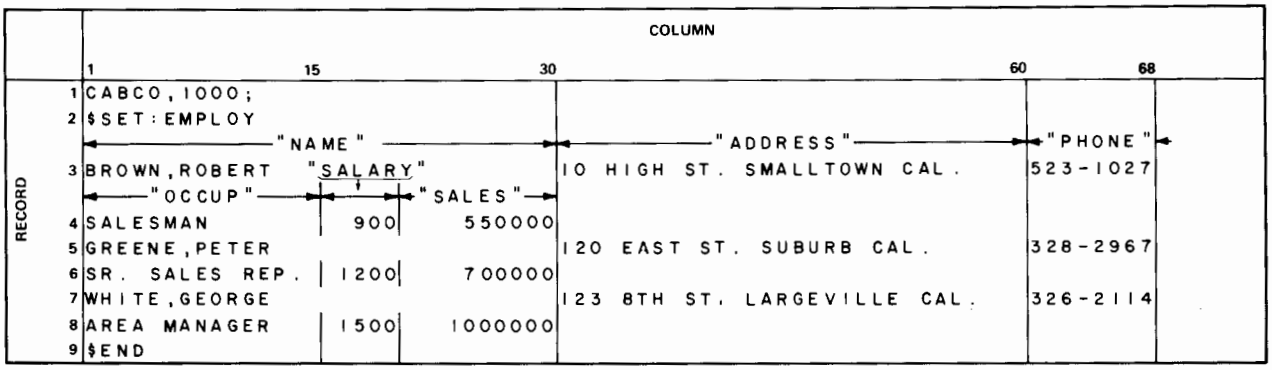


Figure 5-2. DBBLD Data File

DBULD

DBULD stores a copy of the data base information onto magnetic tape. Only data entries from manual master data sets and from detail data sets are unloaded. The DBULD routine is invoked through a DOS-III directive as follows:

:PROG,DBULD,p_1

where:

p_1 is the logical unit number of the magnetic tape device.

Once DBULD is invoked, DBULD asks a series of questions to identify the data base to be unloaded and to verify that the user has the proper security code and level word to access the information in the data base. First, DBULD asks for the data base name. The user types in the DOS-III file name containing the data base *root file*. Then DBULD asks for the security code. The user types the integer number defined in the data base schema as the security code for the data base. Finally, DBULD asks for the level 15 code word. The user either responds by typing the level 15 word for the data base or by pressing any key (followed by *return* and *linefeed*) if no level 15 word was defined in the data base schema. For example,

```
:PROG,DBULD,7

DATA BASE NAME?
   SCHOL
DATA BASE SECURITY CODE?
   12345
DATA BASE LEVEL WORD?
   PRESDT
```

Magnetic Tape Format

DBULD unloads only the data from manual master and detail data sets, in that order. DBULD also inserts header information in appropriate places to identify the data and to supply cross-checks for routines using the magnetic tape to load the data base information back into a DOS-III file. There are three types of headers: tape headers, file headers, and data headers.

TAPE HEADER. The tape header appears only once at the beginning of the magnetic tape to identify the tape as an IMAGE/2100 data base unloaded by the DBULD routine. The header (shown in Figure 5-3) consists of 20 16-bit computer words containing the following information:

Words	Contents
0-4	Contain the character string DBLOAD21ØØ, two ASCII characters per word.
5-7	Contain the 5-character root file name as defined in the data base schema, two characters per word. The last half of the third word contains a blank.

Words	Contents
8-10	Contain the EFMP pack number as defined in the data base schema, two ASCII characters per word. The pack number starts with the letters PN, followed by three digits chosen by the user in the data base schema. The last half of the third word contains a blank.
11	Contains the security code as defined in the data base schema. The security code consists of an integer from 1 to 32767.
12	Contains the number of detail and manual master data sets in the entire data base to be unloaded onto the magnetic tape.
13-19	Contain all binary zeroes.

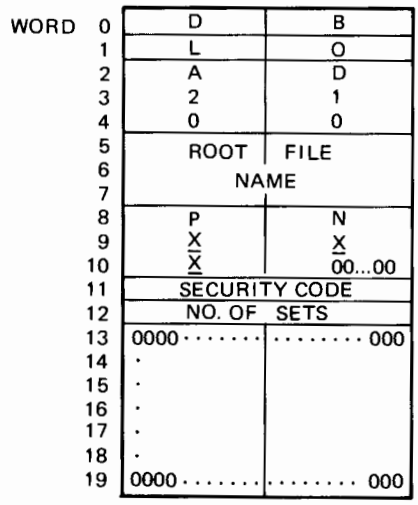


Figure 5-3. DBULD Tape Header

FILE HEADER. DBULD writes a file header on the magnetic tape before every data set. The header contains the ordinal number of the data set (the order in which the data set is unloaded), the name of the data set and the length (in words) of the data set entries. The header (shown in Figure 5-4) consists of 20 16-bit computer words as follows:

Words	Contents
0-3	Contain the character string FILEHEAD, two ASCII characters per word.
4-6	Contain the data set name of up to five characters, two ASCII characters per word. The last half of the third word contains a blank.
7	Contains the tape file number. The first data set stored on tape is file number 1, the second is number 2, and so on.
8	Contains the length (in words) of the data entries in the set.
9-19	Contains all binary zeroes.

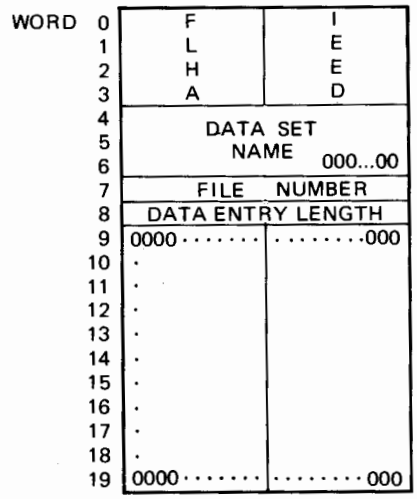


Figure 5-4. File Header

DATA HEADER. Actual data entries contained in a data set are stored in 1024-word blocks identified by a data header. A data block (shown in Figure 5-5) consists of 1024 16-bit computer words as follows:

Word	Contents
0-3	Contain the character string DATAHEAD, two ASCII characters per word.
4	Contains the data block number. The first data block for each data set is block number 1, the second block is block two, and so on.
5	Contains all binary one's to indicate that this block of data is the last block for the current data set. If the block is not the last block in the data set, word five contains all zeroes.
6	Contains the total number of entries loaded from the data set. This word is set to zero for all data blocks in a data set except the last data block of the data set.
7-19	Contains all zeros.
20-1023	Contains the actual data entries contained in the data set. When the 1024-word block is filled, DBULD writes another data header and fills the next block.

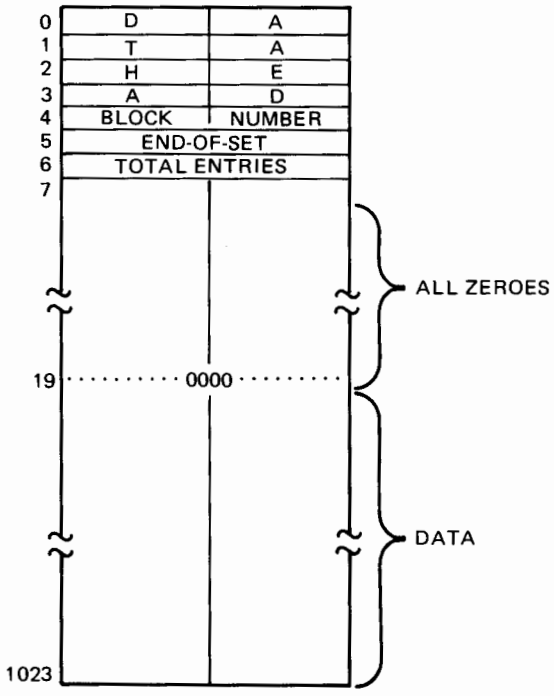


Figure 5-5. Data Block with Header

DBLOD

DBLOD loads data base values into a DOS-III EFMP file (pack number) from a magnetic tape created by the DBULD routine. The data base information can be loaded into the same data base structure, or into a different data base structure. The user of DBLOD accomplishes this by creating a different data base schema. The user can modify the data base schema in the following ways:

- Rename the data base by inserting a new data base name in the data schema.
- Assign a different EFMP pack number by changing the number (PN*xxx*) in the schema. (This EFMP area must be created prior to DBLOD execution through a DOS-III :STORE,B directive.)
- Assign a different security code by changing the integer number used as the security code in the data schema.
- Define a new level structure by modifying the level numbers and corresponding level words in the *level part* of the data schema.
- Change the read and write level numbers for the items in the *item part* of the data schema.
- Define a new item structure for the items in a detail or manual master data set entry. Redefining the structure of a data entry does not alter the physical contents of that entry. Integer data items in the entry must be redefined as integer, real data items must be redefined as real, the character items must be redefined as character. Several character items can be combined into one larger character item if the items exist next to each other in the physical data entry. For example, if two items called NAME and ADRESS are each 30 characters long and exist next to each other in a data entry, they can be combined to form one data item 60 characters long names, for example, NAMADD. Conversly, NAMADD can be broken into two 30-character items called NAME and ADRESS. If the length of a newly-defined entry is shorter than the length of the data entry values on tape, DBLOD truncates the data on tape from the right and the data is lost. If the length of the newly-defined data entry is longer than the length of the values, the unused portion of the defined data entry is padded with binary zeros on the right.
- Change the capacity of a data set to increase or decrease the number of data entries allowed in the data set.
- Add or delete detail data sets. To add a detail data set to the data base, define a data set in the data schema. This set must have a name not contained on the magnetic tape used to load the data base. No data at all will be loaded into this data set by DBLOD. To delete a data set from the data base, omit the data set name on tape from the new data schema. When DBLOD loads data into the data base, the routine scans the schema to find a data set name matching the names on the tape. If no data set name is found in the schema, data for that set is not loaded and is lost to the user. To load data set values into the data base, the same data set name appearing on the tape must appear in the new data base schema.
- Delete manual master data sets by removing the name of the set from the new data base schema. If DBLOD does not find a set name in the schema corresponding to the set name on the tape, data for that set is not loaded and is lost to the user. Data for manual master data sets already existing on the tape can be reloaded merely by using that set name in the new data schema. A manual master data set cannot be defined with a name that did not appear in the old data base schema unless the manual master is not associated with any detail data set.

- Add automatic master data sets by defining new sets in the data base schema. If a new automatic master data set is created, care must be taken that key items in detail data sets reference the proper set name. DBLOD creates the necessary links and inserts the necessary item values into the automatic master data set key items at the time the data base is loaded by DBLOD.

- Redefine key items in detail data sets by changing the reference to master data sets. If key items in detail data sets are changed, care must be taken that key item values contained in a detail data set already exist in the manual master data set key items.

DBLOD Operation

Before using the DBLOD routine, the user first purges and then restores the EFMP data base file (PN*xxx*) using the DOS-III :PU and :STORE,B directives. Then the user invokes the DBLOD routine by typing:

:PROG,DBLOD,p_1

where

p_1 is the logical unit number of the magnetic tape unit containing the data base information to be loaded by DBLOD.

Once invoked, DBLOD asks a series of questions to identify the data base to be loaded and to verify that the user has the proper security code and level word to load data into the data base structure. First, DBLOD asks for the name of the data base to receive the data. The user responds by typing the DOS-III file name containing the data base *root file*. (This is the same name that appears in the BEGIN DATA BASE *name* record in the data schema.) Next, DBLOD asks for the security code of the data base. The user types the integer number defined in the data base schema as the security code for the new data base. Finally, DBLOD asks for the level 15 code word as defined in the data base schema. The user either responds by typing the level 15 word for the data base or by pressing any key (followed by return and linefeed) if no level 15 word was defined in the data base schema. For example:

```
:PROG,DBLOD,7

DATA BASE NAME?
   SCHOL
DATA BASE SECURITY CODE?
   12345
DATA BASE LEVEL WORD?
   PRESDT
```

DBSTR

The DBSTR routine dumps a data base root file and associated detail, automatic and manual master data sets, (including pointers) to a magnetic tape, sector by sector. To invoke DBSTR, the user enters the DOS-M directive:

:PROG,DBSTR,p_1

where

p_1 is the logical unit number of the magnetic tape drive to which DBSTR dumps the root file and associated data.

Once invoked, DBSTR asks a series of questions to identify the data base to be dumped and to make sure that the user possesses the proper security code and level word to access the data base. First, DBSTR asks for the data base name. The user responds with the name of the DOS-III file containing the data base root file. Then DBSTR asks for the data base security code. The user types the integer number defined as the data base security code in the data base schema. Finally, DBSTR asks for the level 15 code word. The user responds by typing the level 15 word for the data base or by pressing any key, followed by *return* and *linefeed*, if no level 15 word was defined in the data base schema. For example:

```
DATA BASE NAME?
   CABCO
DATA BASE SECURITY CODE?
   12345
DATA BASE LEVEL WORD?
   ADMIN
```

Once the user correctly answers the three questions, DBSTR dumps the root file and data base to the magnetic tape unit specified by p_1.

Magnetic Tape Format

DBSTR dumps the entire data base root file and data base entries to magnetic tape, along with tape header information to identify the data and provide cross-checking information for the DBRST routine (which loads the magnetic tape created by DBSTR back into DOS-III files). There are two types of headers: DBSTR tape header and DBSTR data header.

DBSTR TAPE HEADER. The tape header appears only once at the beginning of the magnetic tape to identify the tape as an IMAGE/2100 data base unloaded by the DBSTR routine. The header (shown in Figure 5-6) consists of 20 16-bit computer words containing the following information:

Words	Contents
0–5	Contain the character string DBSTOREΔ2100 (Δ = blank), two ASCII characters per word. The last half of word number 3 contains a blank.
6	Contains the size (in sectors) of the data base root file.
7	Contains the size (in sectors) of the data base (residing on the EFMP file).
8–19	Contains all zeroes.
20–1299	Contains the root file, stored sector by sector.

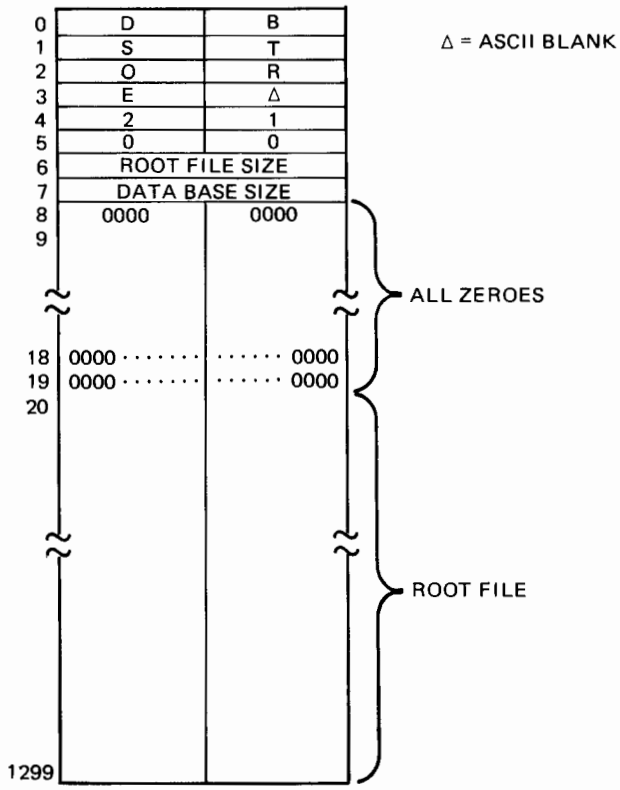


Figure 5-6. DBSTR Tape Header

DBSTR DATA HEADER. Actual data entries in the data base are stored in 1300 word blocks identified by a 20-word data header. A data block (shown in Figure 5-7) consists of 1300 16-bit computer words as follows:

Words	Contents
0–3	Contain the character string DATAHEAD, two ASCII characters per word.
4	Contains the block count. The first block of data is block 1, the second is block 2, and so on.
5	Contains the number of sectors in the data block.
6–19	Contains all zeroes.
20–1299	Contains the sectors of the data base.

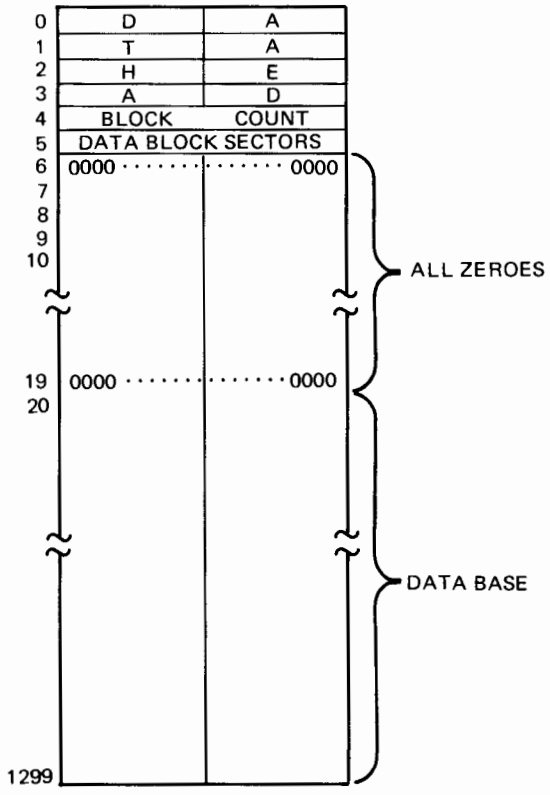


Figure 5-7. DBSTR Data Header

DBRST

The DBRST routine loads a data base and associated root file into DOS-III files from a magnetic tape created previously by the DBSTR routine. No modification of the root file is allowed, and the magnetic tape used must have been created by the DBSTR routine only. To invoke DBRST, the user enters the DOS-III directive:

:PROG,DBRST,p_1

where

p_1 is the logical unit number of the magnetic tape drive containing the tape to be loaded into the DOS-III files.

Once invoked, DBRST asks for a series of questions to identify the data base to be loaded and to make sure that the user possesses the proper security codes to access the data base. First, DBRST asks for the data base name. The user responds with the name of the DOS-III file to contain the data base root file. Then, DBRST asks for the data base security code. The user types the integer number defined as the data base security code in the data base schema. Finally, DBRST asks for the level 15 code word. The user responds by typing the level 15 code word for the data base by pressing any key (followed by *return* and *linefeed*) if no level 15 word was defined in the data base schema. For example:

```
DATA BASE NAME?
   CABCO
DATA BASE SECURITY CODE?
   12345
DATA BASE LEVEL WORD?
   ADMIN
```

Once the user correctly answers the three questions, DBRST loads the data base and root file to the files specified by the data base name and the EFMP pack number defined in the data base schema.

If the user dumps the data base to magnetic tape using DBSTR and then purges the data base file (PN*xxx*), the user must re-create the file (prior to using DBRST to restore the data base) using the DOS-III *:STORE,A, PNxxx,n* directive. If the root file is purged, it must be re-created using the DOS-III *:STORE,B, name,n* directive.

APPENDIX A
Sample Data Schema

A taxi company wishes to establish a data base with several different data sets. The supervisors need a data set to control the scheduling of shifts and the assignment of drivers to cabs in the fleet. Personnel wants an employee file for information about the drivers, while the maintenance shop desires information about the condition of cabs that they maintain. Figure A-1 shows the data base schema (written in DBDL) created by the data base designer. The data base consists of a detail data set and two master data sets.

The detail data set (called SCHEDUL) holds the driver's name, the shop number of the cab assigned to the driver, the area in which the driver will operate the cab, the driver's supervisor, and the hours (shift) the driver works. The master data set (called DRIVER) contains the name, address, phone number, city license number, age and date of hire for all the drivers in the company. The master data set (called AUTO) consists of the company show number, the automobile serial number, license number, date of purchase, manufacturer, the model, the price of all the cabs in the company's fleet. The detail data set (SCHEDUL) is linked to DRIVER through the key item named OPERTR, and is linked to AUTO through the key item named CABNO. The key items link the detail set with two separate master sets to speed information retrieval time when the IMAGE/2100 QUERY subsystem is used to enter and retrieve data.

The master data sets DRIVER and AUTO are both type MANUAL so that more than one data item can be included in each data entry (AUTOMATIC masters allow only one item per entry), and so that a key item value (such as the name of a driver or the shop number of a cab) is entered into the detail data set only once.

In order to enter a data entry into the detail data set, the key item value must first be entered in the appropriate master data set entry. If that particular key item value already exists in the master set, the user is so informed. This prevents entering the same cab number twice in the master data set, or entering the same driver name twice in the other master data set. This also prevents assigning a driver to a cab prior to entering his name in the manual master set.

In order to process the data schema through DBDS, the data base designer performs the following steps:

1. User prepares space for creation of a *root file* by DBDS by entering the DOS-III :STORE,B directive ..... ST,B,CABCO,2.
2. User starts DBDS execution by entering the DOS-III directive :PROG,DBDS,*logical unit* where logical unit 7 is the card reader containing the ASCII schema file ..... :PROG,DBDS,7
3. DBDS reads the schema from the card reader and produces the output shown in Figure A-1.

```
          HEWLETT-PACKARD IMAGE/2100 DATA BASE DEFINITION PROCESSOR

$CONTROL   TABLE,ERRORS=50,NOROOT;
EFMP PACK  ID PN001;100;
BEGIN DATA BASE CARCO;
LEVELS:
       1        CLERK;
       4        SUPERV;
       8        MANAGE;
       15       OWNER;
ITEMS:
       CARNO,   I1    (1,15);   <<NUMBER ASSIGNED TO CAR>>
       OPERTR,  U30   (1,15);   <<KEY ITEM LINKED TO "DRIVER">>
       AREA,    U4    (1,1);    <<AREA OF CITY ASSIGNED TO DRIVER>>
       SUPERV,  U16   (1,1);    <<SUPERVISOR IN CHARGE OR DRIVER>>
       SHIFT,   I1    (1,1);    <<1=DAY,2=NITE,3=GRAVEYARD>>
       NAME,    U30   (4,15);   <<KEY ITEM**SAME AS OPERTR>>
       ADDRES,  U60   (4,4);    <<DRIVER'S HOME ADDRESS>>
       PHONE,   U8    (4,4);    <<DRIVER'S HOME PHONE   XXX*XXXX>>
       LICENS,  I1    (4,4);    <<DRIVER'S CITY CAR LICENSE #>>
       AGE,     I1    (4,15);   <<AGE IS ACCESSED BY OWNER ONLY>>
       HIRE-D,  U6    (4,4);    <<DATE DRIVER IS HIRED BY COMPANY>>
       CAR,     I1    (1,15);   <<KEY ITEM FOR "AUTO"-SAME AS CARNO>>
       SERIAL,  U20   (1,1);    <<CAR SERIAL NUMBER FROM MAKER>>
       PLATES,  U6    (1,1);    <<LICENSE PLATE NO. OF CAR>>
       P-DATE,  U6    (1,1);    <<DATE OF PURCHASE FROM MAKER>>
       MAKER,   U20   (1,1);    <<CAR MANUFACTURER>>
       MODEL,   U20   (1,1);    <<MODEL OF CAR--LIMO.FOUR-DOOR,ETC.>>
       PRICE,   I1    (1,1);    <<PRICE OF CAR WHEN PURCHASED>>
SETS:
<<MASTER DATA SETS>>
NAME:DRIVER,M;      <<THIS MASTER CONTAINS INFO ABOUT DRIVERS>>
ENTRY:
       NAME(1),     <<KEY ITEM LINKED TO "SCEDUL" DETAIL SET>>
       ADDRES,
       PHONE,
       LICENS,
       AGE,
       HIRE-D;
CAPACITY:  20;
NAME:AUTO,M;        <<THIS MASTER CONTAINS INFO ABOUT THE FLEET>>
ENTRY:
       CAR(1),      <<KEY ITEM LINKING TO "SCEDUL" DETAIL SET>>
       SERIAL,
       PLATES,
       P-DATE,
       MAKER,
       MODEL,
       PRICE;
CAPACITY:  20;
```

Figure A-1. Sample Data Schema

```
<<DETAIL DATA SET>>
NAME:SCEDUL,D;      <<SHOWS CAR & DRIVER ASSIGNMENTS FOR SCHEDULING>>
ENTRY:
        CARNO(AUTO),     <<KEY ITEM LINKING "SCEDUL" WITH "AUTO">>
        OPERTR(DRIVER), <<KEY ITEM LINKING "SCEDUL" WITH "DRIVER">>
        AREA,
        SUPERV,
        SHIFT;
CAPACITY:  20;
END.
```

DATA SET NAME	TYPE	FLD CNT	PATH CNT	ENTR LGTH	MED REC	CAPAC CT
DRIVER	M	6	1	54	6	20
AUTO	M	7	1	38	6	20
SCEDUL	D	5	2	27	5	20

Figure A-1. Sample Data Schema (cont.)

APPENDIX B
DBDS Messages

DBDS prints two kinds of messages during DBDS execution — error messages (which occur when DBDS discovers an incorrect schema statement) and general messages (which DBDS prints after the schema is processed).

ERROR MESSAGES

Whenever DBDS encounters an error in a schema statement, DBDS prints

**ERROR:

followed by the appropriate error message as listed in Table B-1. If the $CONTROL "list" option is active (see Section III), DBDS prints the message following the offending statement as part of the data schema listing. If the "no list" option is active, DBDS prints the offending statement followed by the error message. If DBDS discovers an error while processing the schema *set part*, DBDS ignores subsequent characters until the next NAME: or END. statement is encountered. If DBDS discovers an error in any other part of the schema, DBDS ignores subsequent characters until a semi-colon (;) or the word LEVELS: , ITEMS: , SETS: , NAME: , or END. is encountered.

If a statement terminator such as a semi-colon or a list separator such as a comma is missing, DBDS may ignore an entire schema statement, causing subsequent error messages to occur. For example, Figure B-1 shows part of a data schema with a semi-colon missing after the "SALARY, I1(1,6)" statement. Omission of the semi-colon causes DBDS to ignore the following characters up to the next semi-colon which means that DBDS completely ignores the "NAME,U20(1,6)" statement. This causes a subsequent error message to occur (as shown in Figure B-1).

```
          HEWLETT-PACKARD IMAGE/2100 DATA BASE DEFINITION PROCESSOR

$CONTROL LIST,TABLE,NOROOT,ERRORS=5Ø;
EFMP PACK ID PNØØ1;1ØØ;
BEGIN DATA BASE SAMBO;
LEVELS:
      1   CLERK;
      4   MANAGE;
      15  PRESOT;
ITEMS:
      W**NAME,U2Ø(1,6);
      C**EYES,U2(1,6);
      D**HAIR,U6(1,6);
      E**NUMB,I1(1,6);
       SALARY,I1(1,6)
      NAME1,U2Ø(1,6);
***ERROR: BAD TERMINATOR - 1 ; 1 EXPECTED.
      EYES1,U2(1,6);
      HAIR1,U6(1,6);
      NUMB1,I1(1,6);
      PAY1,I1(1,6);
SETS:
                              <<MASTER DATA SETS>>
      NAME: NAME,A;
      ENTRY:
      NAME1(1);
***ERROR:  UNDEFINED ITEM REFERENCED.
      CAPACITY: 1ØØ;
```

Figure B-1. Sample Error Messages

Table B-1. DBDS Error Messages

Message	Description
AUTOMATIC MASTER MUST HAVE KEY ITEM ONLY	Master data sets of type AUTOMATIC must contain data entries with only one data item, and that data item must be defined as the key item.
BAD CAPACITY COUNT OR TERMINATOR	The capacity count in the CAPACITY: statement is not an integer or not between 1 and 32,767 inclusive.
BAD DATA BASE NAME OR TERMINATOR	The *data base name* in the BEGIN DATA BASE *data base name* statement is not a valid DOS-III file name of five characters or less.
BAD LEVEL NUMBER OR TERMINATOR	A *level number* in the *level part* of the schema is not an integer between 1 and 15, is not in ascending order in the list, or is not 15 (if the number is the last number in the list).
BAD LEVEL WORD OR TERMINATOR	A *level word* in the *level part* of the data schema contains an illegal character, or is more than six characters long, or is not ended by a semi-colon.
BAD PACK NUMBER	The pack number must be PN followed by three digits from 001 through 999, inclusive.
BAD PATH COUNT OR TERMINATOR	The path count for a *MANUAL* master data set item is not an integer between 0 and 5. The *path count* for an AUTOMATIC master data set item is not an integer between 1 and 5, inclusive.
BAD READ LEVEL OR TERMINATOR	The *read level* for an item in the schema *item part* is not an integer from 1 to 15, inclusive.
BAD SET NAME OR TERMINATOR	The *setname* for a master or detail data set does not start with a letter, or is more than six characters long, or is not unique (in the first five characters), or contains an ASCII character other than those allowed for *setnames.*
BAD SET NAME OR TERMINATOR IN REFERENCE	The *setname* enclosed in parentheses, following the item name in a detail data set definition, does not follow the rules of setnames.
BAD TERMINATOR–';' EXPECTED	A semi-colon is missing at the end of a schema statement.
BAD TERMINATOR–',' OR ';' EXPECTED	A comma or semi-colon is missing from a schema statement.
BAD TYPE DESIGNATOR	In the *set part* of the data schema, the data set *type* must be MANUAL (or M), AUTOMATIC (or A), or DETAIL (or D).
BAD WRITE LEVEL OR TERMINATOR	The *write level* for an item in the schema *item part* is not an integer from 1 to 15, inclusive.

Table B-1. DBDS Error Messages (cont.)

Message	Description
'BEGIN DATA BASE' EXPECTED	The second record of a data schema (not counting a $CONTROL record) must be BEGIN DATA BASE *data base name.*
'CAPACITY:' EXPECTED	Each data set defined in the schema *set part* must contain a CAPACITY: statement.
DATA BASE HAS NO DATA SETS	A data schema must define at least one data set in the *set part* of the schema. DBDS execution terminates.
DUPLICATE ITEM NAME	An item name must appear once and only once in the data schema *item part* and appear once and only once in the schema *set part.*
DUPLICATE SET NAME	Each set in the data base must be defined with a unique *set name.* (The *set name* must be unique through the first five characters — the sixth character is ignored by DBDS.)
DUPLICATE SET NAME IN REFERENCE	The master data set name appearing as a *link* in a detail data set definition has already appeared once in the definition of the detail data set. A detail data set can be linked to a specific master data set only once.
'EFMP PACK ID' EXPECTED	The first statement of a data schema (not counting a $CONTROL record) must be EFMP PACK ID.... .
'END' FOUND WHERE NOT EXPECTED	An END statement is not the last statement. DBDS terminates processing.
END OF FILE ENCOUNTERED — PROCESSING TERMINATED	DBDS encountered at end of file prior to encountering the "END." record in the schema.
'ENTRY:' EXPECTED	Each data set defined in the schema *set part* must contain an ENTRY: statement followed by item names of data items in the set.
ENTRY TOO BIG	The combined length (in words) of the media record and the data record exceeds 255.
ILLEGAL CONTROL CARD	The $CONTROL record is incorrect.
ILLEGAL ITEM NAME OR TERMINATOR	A data item name is not from one to six characters long (starting with a letter), or contains an illegal character, or is not followed by a comma.
ILLEGAL SECURITY CODE	The *security code* in the first record of the data schema is not an integer between 1 and 32,767 inclusive.
ILLEGAL SPECS	The *spec* in a statement in the *item part* of the schema is not I1, R2, or U *integer,* where *integer* is an even integer not exceeding 126.
ITEM LENGTH NOT INTREGAL WORDS	For the *spec* U *integer, integer* is not even.

Table B-1. DBDS Error Messages (cont.)

Message	Description
ITEM(S) NOT DECLARED IN A DATA SET	Data items appearing in the *item part* of the data schema did not appear under ENTRY: in any data set defined in the schema *set part.*
ITEM SPECIFIED IN PREVIOUS SET	A data item appearing in a data set has been used previously in the set part. An item name defined in the *item part* must appear once and only once within the *set part.*
ITEM TOO LONG	A data item defined as type character (U *integer*) exceeds 63 words in length (*integer* greater than 126).
KEY ITEM DOES NOT HAVE WRITE LEVEL 15	A data item specified as a key item in a data set must be defined with a write level of 15 in the *item part* of the data schema.
KEY ITEMS NOT OF SAME LENGTH	The key item in a detail data set is not the same length as a key item in a master data set linked to the detail.
KEY ITEMS NOT OF SAME TYPE	The key item in a detail data set is not the same type (I1, R2, or U *integer*) as a key item in a master data set linked to the detail.
LEVEL 15 WORD NOT SPECIFIED	A level word corresponding to a level number of 15 must appear in the *level part* of the data schema, if the *level part* exists at all. If the user decides that no security will exist for any data item in the data base, then no level 15 word need be specified.
'LEVELS:' NOT FOUND	Even if no *level part* exists for a data schema, the word "LEVELS:" MUST APPEAR as the third statement in the schema. DBDS execution terminates.
LEVEL WORD TOO LONG	A *level word* appearing in the *level part* of the schema is greater than six characters or contains an illegal character.
MASTER DATA SET LACKS EXPECTED DETAILS	A key item was defined in a master data set by means of a non-empty *path count* linking the master to one or more details. The detail data set(s) were not subsequently defined in the *set part* of the data schema.
MASTER DATA SET LACKS KEY ITEM	A master data set was defined without a key item. (All items in the master set had empty *path counts.*)
MAX ERRORS – SCHEMA PROCESSING TERMINATED	DBDS has discovered the maximum number of errors allowed by the $CONTROL options specified. DBDS terminates schema processing.

Table B-1. DBDS Error Messages (cont.)

Message	Description
MORE THAN ONE KEY ITEM	More than one item in the data set appears with a non-empty path count in a master data set.
'NAME:' OR 'END.' EXPECTED	Either another set definition did not occur after one data set was defined in the schema, or the END. record is missing following the last data set defined in the schema *set part.* DBDS execution may terminate.
REFERENCED SET NOT MASTER	A name used to define a key item in detail data set is not the name of a previously defined master data set.
SCHEMA PROCESSING TERMINATED	DBDS cannot continue processing the data schema.
SET HAS NO PATHS AVAILABLE	More detail data sets have specified links to a master data data set than are specified in the master data set *path count.*
TOO MANY DATA ITEMS	Only 100 data items are allowed in a data schema.
TOO MANY DATA SETS	Only 20 data sets can be defined in a data schema. DBDS execution terminates.
TOO MANY PATHS IN DATA SET	The *path count* in a master data set exceeds 5.
UNDEFINED ITEM REFERENCED	An item name not referenced in the schema *item part* appears in a data set description.
UNDEFINED SET REFERENCED	A master data set name not yet defined in the schema appeared in a detail data set description.

GENERAL MESSAGES

After DBDS processes a data schema, it prints a group of general messages informing the user about the size of the root file, number of errors encountered, etc. Table B-2 lists the messages which can occur at the bottom of the schema listing.

Table B-2. General Messages

Message	Description
DATA BASE LENGTH: *x* SECTORS	Informs the user how many sectors (x) are necessary for the data base. This information is useful for creating the data base file (i.e., :ST,B,PN*nnn*,x reported in the message).
DATA SET COUNT: *x*	Informs the user how many data sets (x) are defined in the data schema.
ITEM NAME COUNT: x	Informs the user how many data items are defined for the data schema.
NOT ENOUGH SECTORS IN ROOT FILE	The root file defined by the user prior to DBDS execution is not large enough to hold the actual root file created by DBDS.
NUMBER OF ERROR MESSAGES: *x*	Informs the user how many error messages were printed by DBDS during schema processing.
ROOT FILE *name* CREATED	Informs the user that DBDS stored the root file in the file space (called *name*) created by the user prior to DBDS execution.
ROOT FILE NOT CREATED DUE TO ERROR(S) IN SCHEMA	If the ROOT control option is active and errors occur in the data schema, DBDS prints this message to inform the user that no root file was created.
ROOT FILE NOT FOUND -- MUST BE DEFINED BY USER	DBDS was unable to find a file whose name matched the name given in the data schema BEGIN DATA BASE *data base name;* statement. The name of the file must match the *data base name* and must be defined by the user prior to DBDS execution using the :STORE,B directive.
ROOT LENGTH: *x* WORDS	Gives the length (in words) of the schema root file. The disc holds 128/words/sector.

APPENDIX C
DBUS Error Messages

If an error occurs during operation of one of the five DBUS routines (DBBLD, DBULD, DBLOD, DBSTR, DBRST), DBUS prints an error number. Listed below are error numbers followed by the error conditions which may cause them.

Error Number	Description
201	Incorrect data base name. The data base name read as part of the input file to DBBLD is too long, or consists of a non-printing ASCII character, or does not contain a proper terminator.
202	Incorrect security code. The security code read as part of the input file to DBBLD does not consist of an integer between 1 and 32767.
203	No data appears in the DBBLD input file between the first record and the $END record. Make sure that the $END record is the last record in the input file.
204	A data entry in the DBBLD input file is too short, or omitted. Make sure that all the data fields for the entry are present and in the proper order. Make sure that the $SET: record is in the proper sequence.
205	A $SET: or $END record is missing, or too many fields exist for a data entry in a DBBLD input file.
206	A non-numeric character appeared in an integer data field in a DBBLD input file. The only exception is negative numbers, which contain an 11-zone character as the last digit of the number. Consequently, numbers such as 100M (the representation of -1004) are legal.
207	A non-numeric character appeared in a real data field in a DBBLD input file. Real values appear without decimal point.
208	An error has occurred during entry into a data base from a DBBLD input file. DBBLD cannot purge the data base. The user must purge the data base using a DOS-III:PU directive, then recreate the data base using a DOS-III:STORE,B directive.

Error Number	Description
209	A physical end-of-file mark was encountered by DBBLD while processing an input file. Check for a missing $END in the last record of the input file.
210	The logical unit number specified in the DOS-III directive to invoke either DBULD, DBLOD, DBSTR, or DBRST is not an integer number between 1 and 63 inclusive.
211	The user has failed to give the correct level 15 word when asked to do so by the DBULD, DBSTR, or DBRST routines.
212	The input tape used for the DBLOD or DBRST routines is faulty or not created by the DBULD or DBSTR routines, respectively.
213	The user has failed to give the proper security code when asked to do so by the DBSTR, or DBRST routine.
214	The root file and associated data base data are on different subchannels during operation of DBSTR or DBRST.
215	Either the file containing the data base root file or the file containing the data base cannot be found during operation of the DBSTR or DBRST routines.
216	Either the root file size (in sectors) or the data base file size on the disc is not the same size as the root file and data base stored on a magnetic tape by the DBSTR routine.
217	The DBULD routine has discovered no data in the data base to unload.

02100-90138