IMAGE/45 Programming

Part No. 09845-91055
Microfiche No. 09845-98055

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

June 1980...First Edition
November 1980...Second Edition. Updated pages: 5, 45, 85, 94, 95

Preface

The IMAGE/45 Programming manual is designed for IMAGE/45 users who intend to write programs to access their data bases. The user of this manual is assumed to be familiar with the material presented in the System 45 Operating and Programming manual, including mass storage concepts. Those users who do not intend to write data base programs should refer to the QUERY/45 User's Guide.

The IMAGE/45 Programming manual is written to serve three functions –

- Introduce the important concepts and terms of data base management, and the statements used to implement them, to beginning IMAGE/45 programmers.
- Provide syntax reference information for all programmers.
- Introduce advanced data base management topics.

Data base management concepts, terms, operations and statements are presented in a topical format. The data base management functions are defining and creating a data base, opening and closing it, storing, retrieving and modifying data and data base maintenance. Chapters 2 through 7 contain this information. Chapter 8 contains the general-purpose statements and functions provided by IMAGE/45.

Reference information is found in two places. Appendix A is a glossary of IMAGE/45 terms while Appendix B contains the syntax of each IMAGE/45 statement and function. This information can also be found in the System 45 Quick Reference.

Advanced data base topics include multiple-volume data bases, memory management, and synonyms. This information is found in Chapter 9.

Data Base Design

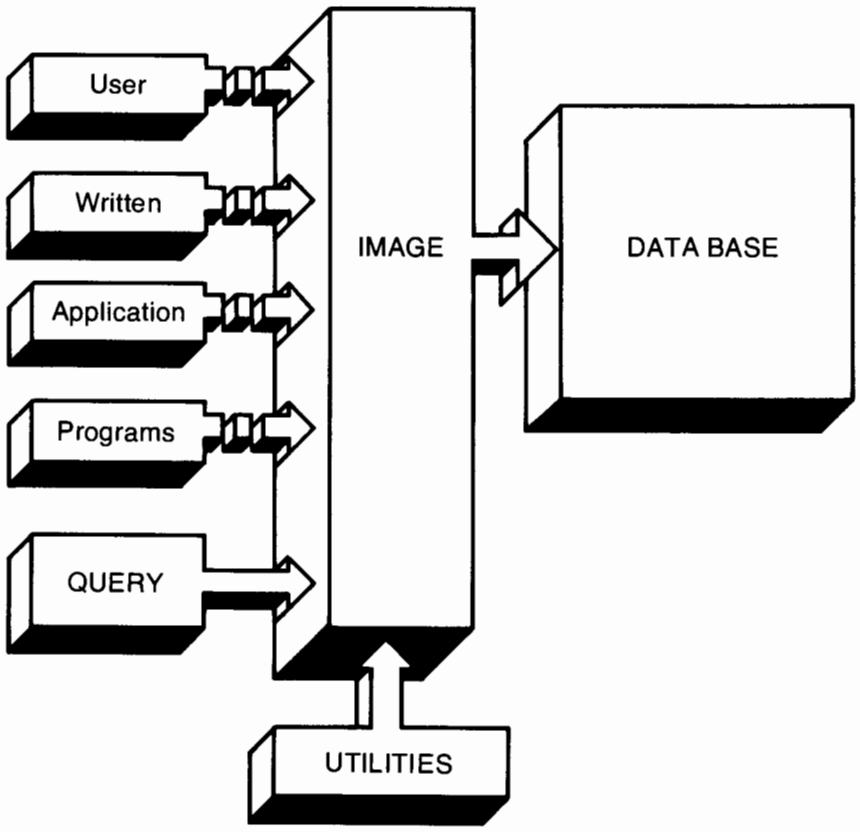
Data base design is an essential step in using a data base management system. A data base should be designed after you are familiar with the information contained in Chapter 2, the terms and concepts of data base management. Since it is an essential step for users of both IMAGE/45 and QUERY/45, the Data Base Design Kit is provided separately for easy access for all users.

Table of Contents

Chapter 1
General Information

Introduction

The IMAGE/45 Data Base Management System (DBMS) provides you with a set of statements and utility programs which enable you to define, access and maintain a data base. A data base is a group of logically related files which contain data and structural information about that data and provides data integrity, independence of data and programs, data security and more efficient data access. A data base management system provides you with tools for organizing and interacting with the data.



This diagram illustrates the IMAGE/45 data base management system. IMAGE/45 integrates the data files for an application and relieves you of the task of data organization and management. QUERY/45 is an interactive program which provides you data base access without having to write a program; it also utilizes IMAGE/45. The Utility programs provide an easy means for maintaining data bases.

Data Base Management Steps

There are six basic procedures and operations that are the necessary steps in data base management.

1. Understand Data Base Concepts and Terms

It is essential to understand the basic concepts and terms in order to use IMAGE/45 effectively. **Chapter 2** is devoted to these ideas.

2. Design the Data Base

Converting your application needs into an optimum design that IMAGE/45 can interpret is the most important step in using your data base management system. The **Data Base Design Kit** provides procedures and tools to make the conversion from need to solution as straightforward as possible.

3. Define the Data Base to IMAGE/45

Once you have designed a data base, it is a simple process to convert your logical design into a **schema**, or formal definition of the data base. The **Schema Processor**, a Utility program, uses the schema text to create the **root file**. The root file, which is the directory to the data base, contains all the structural information about how the data is related. **Chapter 3** discusses the schema.

4. Create the Data Base

In addition to the root file, a data base is comprised of files in which the data is stored. Once the root file is created, the next step is to create these files; this information can also be found in **Chapter 3**.

5. Access the Data Base

Any time you use your data base, it must be **opened**. Data can then be **stored**, **retrieved**, **modified** or **deleted**. After you are done accessing the data base, it should be **closed** to ensure that the root file and the data set files are updated. The statements providing data base access are discussed in **Chapter 4**.

6. Maintain the Data Base

Backing up should be a regular part of your data base operations. **Restructuring** the data base or performing **large-scale erasure** may also be needed from time to time. **Chapters 5, 6 and 7** cover the data base maintenance information.

Advanced Data Base Techniques

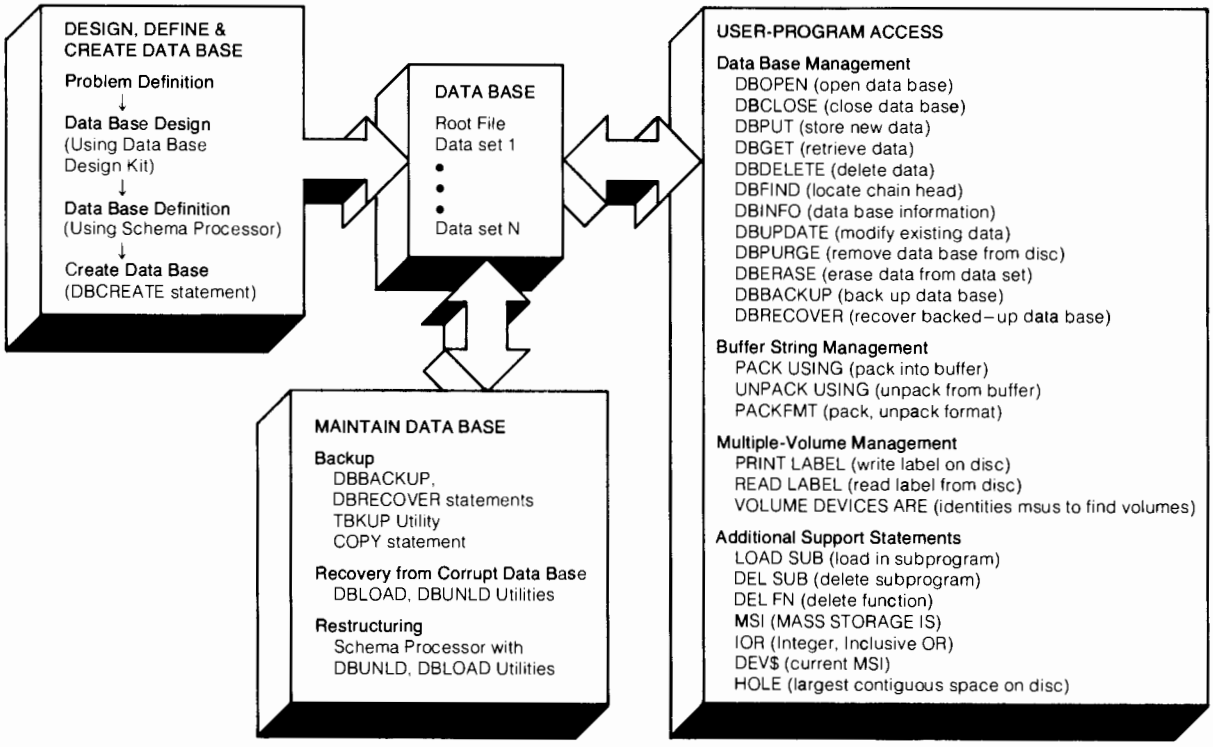
The more advanced uses of IMAGE/45, such as multiple-volume data bases and memory usage, are discussed in **Chapter 9**.

Additional Tools

IMAGE/45 provides you with additional **statements** and **functions** which can be used for many programming tasks. **Chapter 8** discusses these.

Diagram of Data Base Management Steps

The following diagram illustrates the Data Base Management steps just described.



The Sample Data Base

Throughout all of the IMAGE/45 discussions, a sample data base application is used. A copy of this data base (in backed-up form) containing sample data is provided as part of the IMAGE/45 package. It is a simple example and allows you to perform and become familiar with all of the data base steps, from concepts to restructuring, before you create your own.

The sample data base must be transferred from the tape cartridge to a disc and converted from back-up form to the normal configuration. This information can be found in the next section, **Getting Started**.

The Scenario

The NOP Manufacturing Company has eight plants. Each plant has a small library of reference books. The books may be checked out by any employee from any plant and kept as long as needed.

NOP would like to use a System 45 to keep the following information –

- Information about each book (title, author, call number[1], publication date, publisher, cost and subject).
- A list of every book, identifying which plant owns it.
- Whether or not a book has been checked out, and when.
- Which employee has borrowed a book (including name, employee number, department and plant phone number).
- General information about each library branch (plant name and address, librarian and phone number).
- Classification of each book by subject.

Using IMAGE/45, books can be found by title, author, subject or call number. The call number can be used to determine whether or not a book is checked out (and to whom) and if there are any other copies available. Additionally, all books on a particular subject can be found.

General Information

Here is some general information about the sample data base –

Name: LIBR
Password: LIBRMGR
ENGINEER
Maintenance Word: BOOKS

1 A simple numbering scheme is used as a call number to identify each unique Author-Title combination in the library. This scheme is not related to any particular library numbering system.

Getting Started

Before using your data base management system, there are two things you need to do. First, install the IMAGE/45 and Mass Storage ROMs and, if you are using QUERY/45, the Advanced Programming ROM in the computer. The procedure for installing ROMs can be found in the System 45 Owner's/System Exerciser manual. Second, the Utility programs should be backed up and the sample data base must be transferred to a disc.

The COPY statement (discussed in the Operating and Programming manual) can be used to copy the Utility programs from the tape cartridge to another medium. The programs can be copied to a tape or to a disc. It is recommended that the TBKUP program be copied to a disc, since it is used frequently. The following are the file names to COPY –

"TBKUP" "SCHERR"
"SCHEMA" "DBUNLD"
"Confg" "DBLOAD"

NOTE

It is essential that you back up the Utility programs. If the HP-supplied tape failed or was lost, you would have no copy of these programs.

To transfer the sample data base to a disc, the TBKUP Utility program is used to recover backup version, creating the data base files. To recover the data base, follow these steps –

1. Load and run the TBKUP Utility Program (LOAD "TBKUP").
2. Insert the tape cartridge containing the sample data base into one of the tape drives.
3. The program asks you for various information. Enter the following in response to the questions:
   - 2 – to indicate recovery
   - CONTINUE – to indicate a single-volume data base
   - LIBR – the name of the data base
   - LIBRBK – the name of the backup file. Add the mass storage unit specifier –msus– (:T14 or :T15) if the tape is not in the current default mass storage device
   - msus – this is the mass storage unit specifier of the disc on which you want to place the sample data base
   - YES – to indicate there is no previous version of the data base
4. After the root file and data files are recovered, enter YES to recover the QUERY/45 related files.

1 NOTE: The ROM belonging in the righthand drawer should be inserted into one of the two slots closest to the inside of the drawer. The third slot, labeled with a triangle, should actually be labeled with a square.

Equipment Supplied

The IMAGE/45 Data Base Management System consists of the following items. If any items are missing, contact your local Sales and Service Office.

Item	HP Part Number
2 IMAGE/45 ROMs	09845-65526 09845-65527
IMAGE Utilities (tape cartridge)	09845-14757
3 QUERY/45 tape cartridges[1]	09845-14754 09845-14755 09845-14756
Sample Data Base (tape cartridge)	11141-10666
IMAGE/45 Programming Manual	09845-91055
QUERY/45 User's Guide[1]	09845-91056
System 45 Quick Reference	09845-92015
Data Base Design Kit	09845-91057
Binder	9282-0868
2 Error message stickers[2]	7121-0234 (hard errors) 7121-8890 (status errors)
Configuration Letter Registration Reply Card Certificate of Copyrights	09845-91058

Syntax Conventions

The following conventions are used in the statement descriptions found in this manual.

dot matrix – All items in dot matrix must appear exactly as shown.

[ ] – Items within solid square brackets are optional. When several items are arranged vertically within enlongated brackets, one at most may be included. For example, given [A / B], A, B or nothing may be included. Brackets of dot matrix are part of the statement.

... – Three dots indicate that the previous item can be repeated.

| – A vertical line between two parameters means "or"; only one of the two parameters can be included.

System Configuration

To use the IMAGE/45 Data Base Management system, you need –

- System 45 with 187K read/write memory
- Mass Storage ROM
- IMAGE/45 ROMs
- A flexible or hard disc drive with the appropriate interface

1 Not included if 98429A is purchased.

2 The error stickers should be placed on the pull-out card under the CRT which is for option errors.

Chapter 2
Data Base Concepts

Introduction

A data base management system lets you represent, relate and store data on a disc without worrying about how it is organized and maintained. You can keep information about things and how they are related, store this information and retrieve it later. Those of you familiar with other data base management systems may know that there are three primary ways that a DBMS can organize data: hierarchical, relational and network; IMAGE/45 uses a simple network organization.

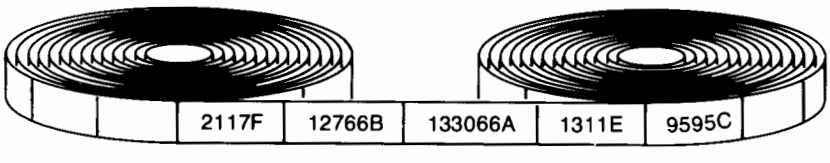
In order to use IMAGE/45, it is essential that you understand how data is represented, related and stored. This chapter begins with a more familiar subject, file management, then moves on to the methods by which IMAGE/45 represents, relates, stores and retrieves data.

File Management

Data can be stored on a mass storage medium for later use. You should already be familiar with CREATE, ASSIGN, PRINT#, and READ#, etc., the data storage capability covered in the System 45 Operating and Programming manual. Data base management is a more highly evolved system of file and data management. This section covers various methods of accessing stored data, providing information which will help you understand how data is stored in a data base.

Serial Access

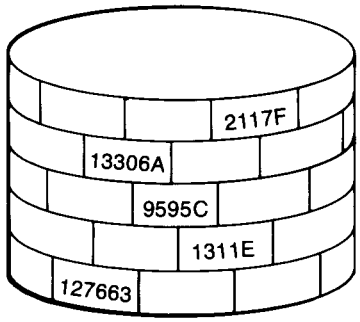
The most basic method of storing and retrieving data is serial access, which is available on the System 45. A tape cartridge is a good example of a serial-access storage medium. Data can be accessed only by accessing the data before it.



Serial Access

Random Access

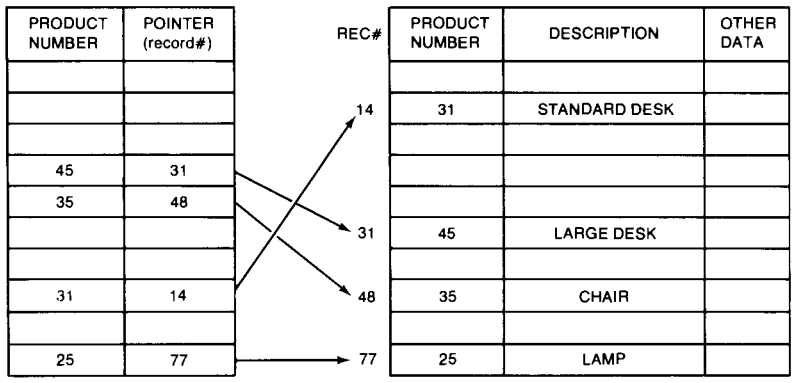
Random access, in which a specific record is accessed directly, was an important step in data storage. Random access is also available on the System 45. A record at the end of a file can be accessed as fast as one at the beginning. A disc is an example of a random-access storage medium.



Random Access

Keyed (Indexed) Access

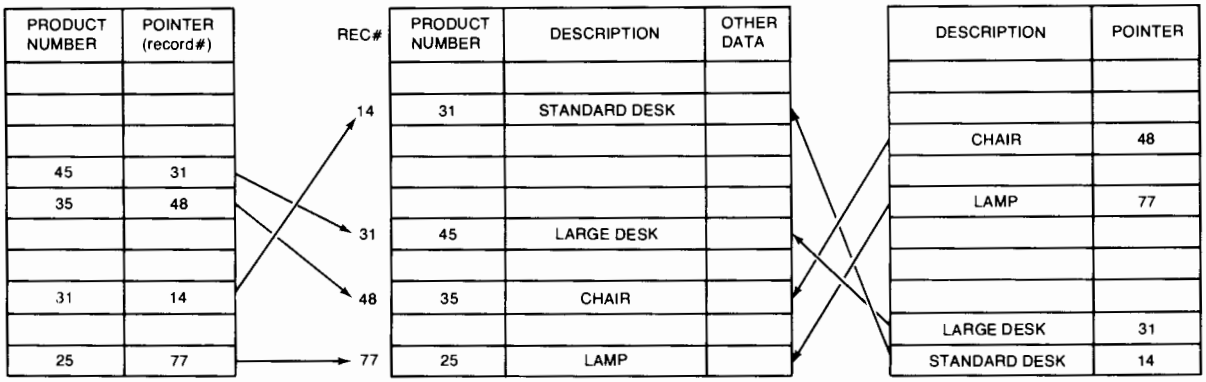
Random access allows any record to be accessed in an equal amount of time. However, it is not useful for finding particular data if its record number is **not** known. In this case, an **index file** is useful. The index file contains values of a particular item of data with record numbers serving as pointers to the corresponding values in the data file. The indexed values (called **keys**) are arranged in some order, normally alphabetical or numerical. A binary search can be used in the index file to find the location in the data file of the desired data.



Keyed Access

Multiple-keyed Access

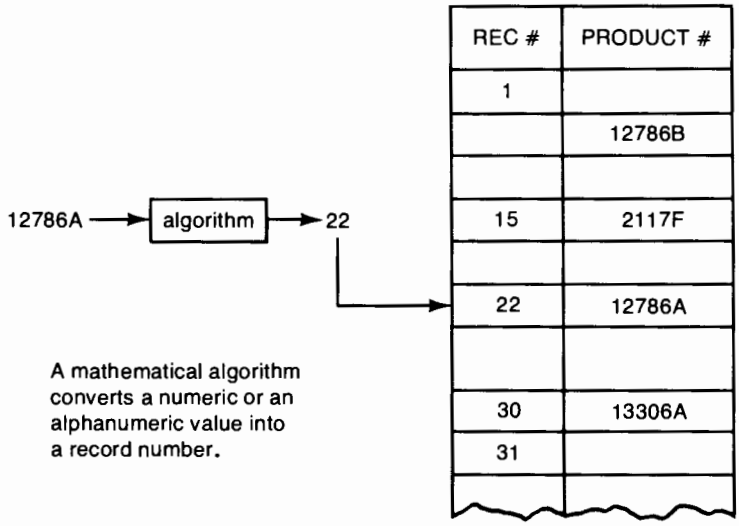
In multiple-keyed access, more than one of the data items in the data file serves as a key. There is a corresponding index file for each key item.



Multiple-keyed Access

Hashing

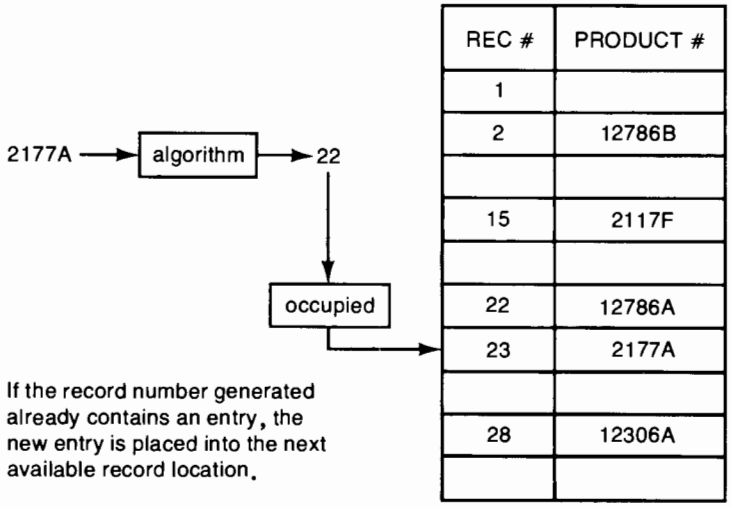
A binary search of an index file can be relatively slow; as an alternative, a process called **hashing** can be used. Hashing manipulates a key value using some algorithm and produces a record number. The algorithm varies with the size of the data file, since the result must be between 1 and the number of records in the file. It also varies with the nature of the key; a numeric value and an alphabetic value require different algorithms.



Hashing

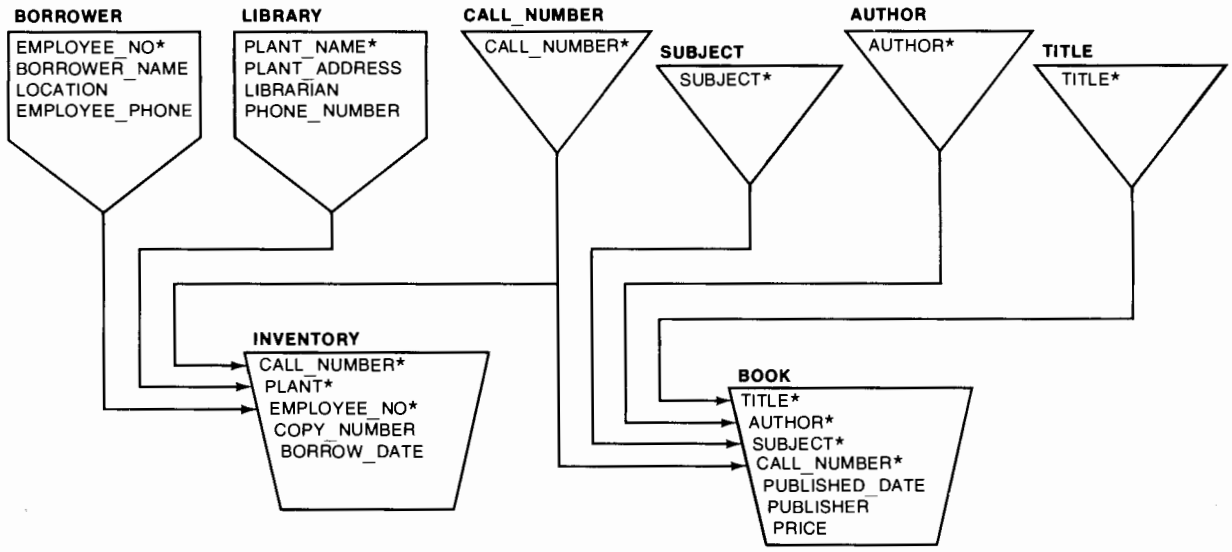
Collisions

Sometimes, hashing different key values produces the same record number; this is called a **collision**. When this happens, the second value is called a **synonym** and is placed in the next available record after the one generated. A pointer is placed in the first record to identify the location of the synonym. Excessive synonyms cause greater access times.



Collision

The record to which a key value hashes is known as its **primary address**. When a collision occurs and a key value cannot be placed at its primary location, it is placed at a secondary location. If a key value is in a secondary location and another key value hashes to that location, the key which is in a secondary location is moved and the new key value is placed in its primary location.


BORROWER
EMPLOYEE_NO*
BORROWER_NAME
LOCATION
EMPLOYEE_PHONE
LIBRARY
PLANT_NAME*
PLANT_ADDRESS
LIBRARIAN
PHONE_NUMBER
CALL_NUMBER
CALL_NUMBER*
SUBJECT
SUBJECT*
AUTHOR
AUTHOR*
TITLE
TITLE*
INVENTORY
CALL_NUMBER*
PLANT*
EMPLOYEE_NO*
COPY_NUMBER
BORROW_DATE
BOOK
TITLE*
AUTHOR*
SUBJECT*
CALL_NUMBER*
PUBLISHED_DATE
PUBLISHER
PRICE

Representing and Relating Data

The sample data base of the NOP Library is shown in the diagram opposite. The diagram shows how the data is represented and related and is used in the following pages in the explanation of how IMAGE/45 represents and relates data.

Entity

A data base is used to keep information about things, or **entities**, and relationships between things. For example, the sample library data base keeps information about three entities: books, borrowers and library branches. It also keeps information about the relationship between things; the inventory is defined in terms of books, their borrowers and the library branch where they are kept.

Attributes

Each entity is described and defined by its various **attributes**, which are identifying characteristics of an entity. The attributes of book, borrower and library branch are shown in the following table.

Entity	Attribute
Book	Title
	Author
	Call Number
	Subject
	Publisher
	Publication date
	Cost
Borrower	Name
	Employee Number
	Location
	Phone number
Library branch	Plant name
	Plant address
	Librarian
	Phone number

Attribute Values

Each book in the library is identified by giving values to its attributes. The following example shows the attribute values for two books.

Attribute	Attribute Values	
Title	Programming Proverbs	Thin Films
Author	Henry Ledgard	J.M. Poate
Call Number	74122058	77125348
Subject	Programming	Semiconductors
Publisher	Hayden	Wiley
Publication date	February 17, 1975	July 17, 1978
Cost	12.95	11.85

Representing Data in a Data Base

A data base management system provides you with a way to represent entities, relationships between entities, attributes and attribute values. These means are covered in the following sections.

Data Item

Attributes are represented in IMAGE/45 by **data items**, which are the smallest accessible data elements. Data items are given values and can correspond to program variables. Each data base can have up to 255 data items, each of which is referenced by a unique data item name.[1]

Types of Data Items

There are four types of data items used with IMAGE/45[2], which give you a choice of how the data is represented. The four types are –

Type	Range
Character String	A maximum, even length of 2 to 1022 characters
Integer	–32 768 to 32 767
Short numeric	6 significant digits and an exponent in the range –63 to 63
Long Numeric (Real)	12 significant digits and an exponent in the range –99 to 99

Here are some example data items –

Data Item Name	Sample Data Item Values
BORROWER_NAME	Chris Bander Bob Noland Don Audina
AUTHOR	Henry Ledgard J.M. Poate Niklaus Wirth

Compound Data Items

The data items just discussed are **simple data items**. A **compound data item** is a type of data item which is, effectively, a one-dimensional array. Each element in a compound data item is called a **subitem**. For example, the item ADDRESS is a compound item of three 40-character strings (subitems). The first subitem (ADDRESS(1)) is used for the address and street, ADDRESS(2) is for the city and state and ADDRESS(3) is for the zip code.

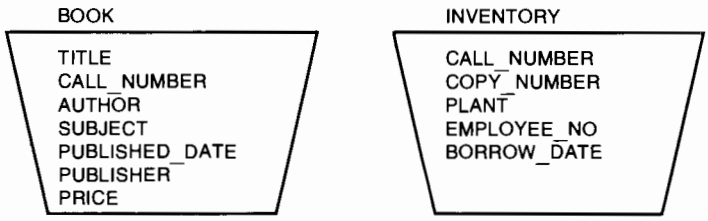
1 With QUERY/45, up to five name synonyms can be specified for a data item name and can be used interchangeably. For example, WRITER and NOVELIST can be name synonyms for AUTHOR.

2 QUERY/45 has three additional data item types: **Name**, a character string used for personal names, **Date**, a special type of short-precision numeric for storing dates, and **Code**, which allows up to 35 values to be specified as the only valid ones for a particular data item and is stored as an integer.

Data Set

An entity is represented in IMAGE/45 by a **data set**. A data set can also represent a connection or relationship between two entities. A data set is a collection of data items that describe the entity or relationship. There can be up to 127 data items in a data set.

Here are two examples of data sets. The BOOK data set keeps information about an entity, a book. The INVENTORY data set keeps information about the relationship between two entities, book and borrower.

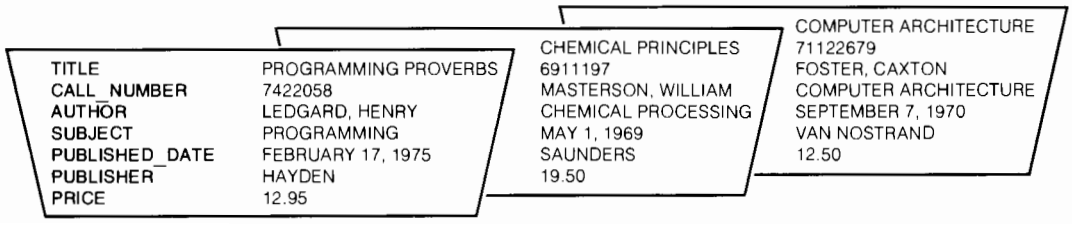


Data Entry

A **data entry**, or **record**, is a collection of values for the items in a data set; these values describe a particular occurrence of the entity or relationship. A data entry can be up to 1 022 bytes in length.

A data set is made up of a number of data entries. When you define a data set, you specify the maximum number of data entries that it can have. The greatest possible number of data entries is 32 767.

Here are three sample entries of the BOOK data set.



Key Items

Normally, each data set has one or more data items by which a data entry is identified or retrieved. These items are called **key** or **search items** and may not be compound items. Key items are indicated with an asterisk (*) in the diagram of the sample data base. For example, a particular book can be identified by its CALL_NUMBER, so this is a key item. Additionally, since one use of the sample data base is to identify books on a particular subject, SUBJECT is also a key item.

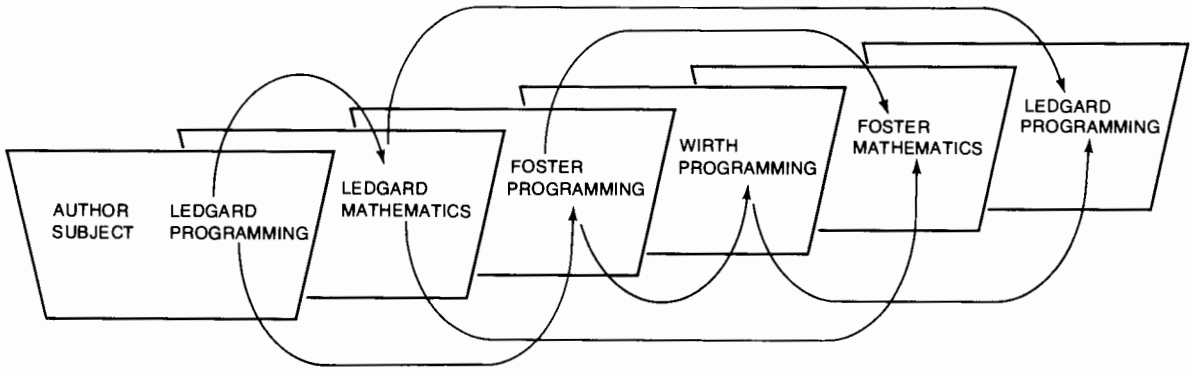
Key items also tie together related data entries which have the same value for the key item. For example, all books which have the same author (AUTHOR is a key item) are automatically linked together.

A third use for key items is to act as a link for information in different data sets. For example, the INVENTORY set contains information about which employee has checked out which books. If you know an employee number for an employee and want to find out the titles of the books he has checked out, the key item CALL_NUMBER is used as a link between the BOOK and INVENTORY data sets.

Data Chain

A data chain is used to link together data entries having identical values for a key item. This makes it easier to retrieve entries which are alike in some way. The first entry in a chain is the **chain head**. IMAGE/45 automatically keeps track of the pointers necessary to link the entries in a chain together.

For example, a data chain links together all the books having the same subject. Another chain links books with the same author.

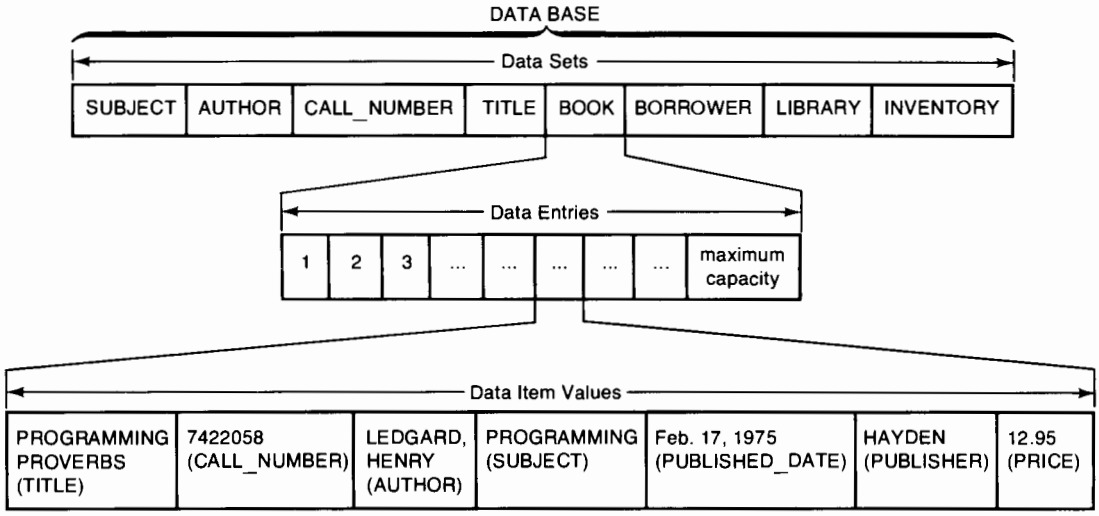


Storing a Data Set

All the entries in a data set are stored together in a separate **data set file** on a disc. Each entry is stored in the file in a unique record known as a **logical record** or **media record**. A media record contains both the data entry and its linkage information, and is given a record number based on its position in the file. Records in a file are numbered consecutively from 1 to the capacity specified when the data base is defined. Logical records that do not contain a data entry are known as empty records.

Data Base

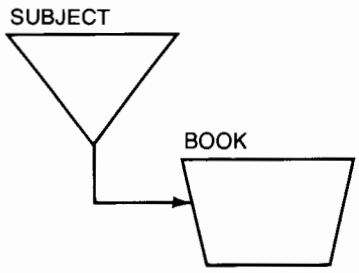
An IMAGE/45 data base consists of one or more data sets, or files, which have some logical relationship with one another. The following diagram demonstrates the structural organization of an IMAGE/45 data base.



Types of Data Sets

There are two types of data sets defined by IMAGE/45: master data sets and detail data sets. Detail data sets contain the bulk of the information. They generally contain data items which describe an entity or relationship between entities. A master data set is a collection of key item values and is generally used as an index for fast access to the information in one or more detail sets. Each master entry contains information about related detail chains, including the beginning and end of and number of entries in the chain.

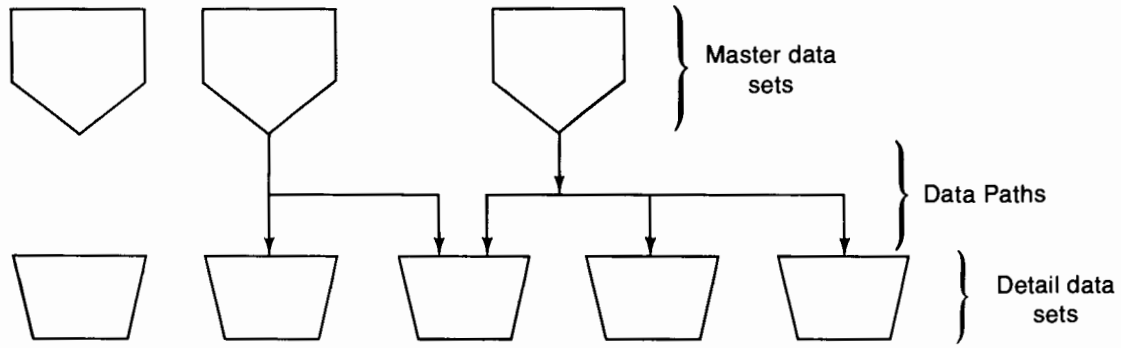
For example, the BOOK detail data set contains information about a book, such as title and subject. A related master set, SUBJECT, is used to find all the books on a particular subject.



Data Path

The connection between a master data set and a detail data set via a key item is known as a **data path**. A detail data set can contain up to 16 key items; each key item defines a path to a master data set. A master data set can have paths to 16 keys in detail data sets.

The diagram below demonstrates how master and detail data sets can be linked together by a variety of paths.



The following example illustrates the difference between paths and chains. The path between SUBJECT and BOOK facilitates retrieval of the first entry in a chain of entries having the same subject. A chain links those entries together.

SUBJECT (master data set)

Record Number	Head of Chain	End of Chain	Chain Count	Key Item SUBJECT
1	2	5	2	MATHEMATICS
2				
3	4	14	3	IND. MANAGEMENT
4	8	8	1	PROGRAMMING
5				
6				
7				
8				
9				
10				

data path

BOOK (detail data set)

Record Number	Forward Chain Pointer	Backward Chain Pointer	Key Item SUBJECT	Other Data Items
1				
2	5	0	MATHEMATICS	
3				
4	9	0	IND. MANAGEMENT	
5	0	2	MATHEMATICS	
6				
7				
8	0	0	PROGRAMMING	
9	14	4	IND. MANAGEMENT	
10				
11				
12				
13				
14	0	9	IND. MANAGEMENT	
15				

data chain

Types of Master Sets

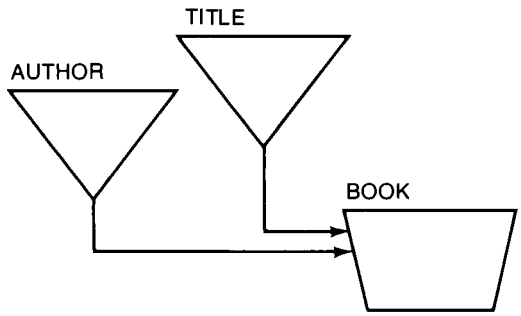
There are two types of master data sets: automatic and manual. An automatic master data set contains one item, the key item. An entry to an automatic master set is made automatically when an entry to a related detail data set is made which contains a new key item value. Similarly, when the last detail entry containing a particular key item value is deleted, the entry in the automatic master is automatically deleted. Thus, an automatic master is useful when numerous additions or deletions make manual updating unreasonable, or when it is unnecessary to check key item values for validity. By necessity, automatic masters must be linked to at least one detail data set.

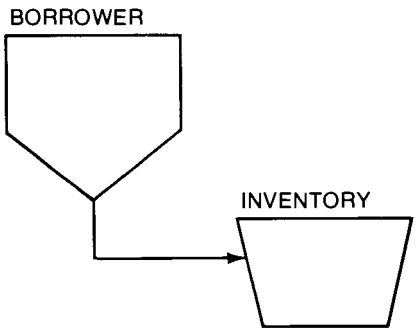
A manual master data set is used to ensure validity of key item values, especially in the related detail sets. Before an entry to a detail set can be made, its key item values must already have been entered explicitly into the related manual master data sets. Similarly, all manual master entries must be manually deleted after all related detail entries are deleted; deleting the last detail entry does **not** delete the entry in the manual master.

A manual master data set can contain items in addition to the key item; this can reduce data repetition by storing unchanging detail information with the key. The BORROWER data set is used in this way. A manual master data set may stand alone; it need not be linked to any detail data set.

Master Set Examples

Books in the BOOK detail set are often accessed by TITLE and by AUTHOR. Since it would be time-consuming to add the TITLE and AUTHOR to the master sets when a new book is added, TITLE and AUTHOR are automatic master data sets.





The BORROWER data set is a manual master to allow other information about a borrower to be kept with the EMPLOYEE_NUMBER, rather than being repeated in INVENTORY entries.

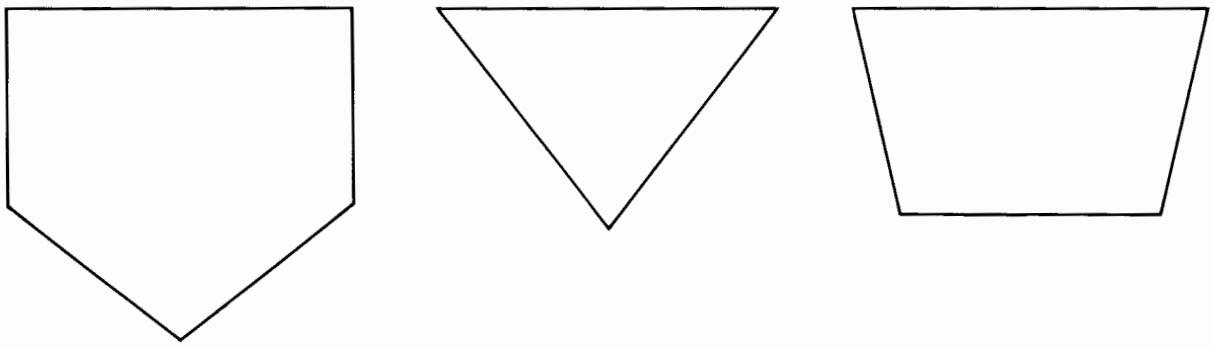
Master Data Set Summary
The following table summarizes the characteristics of manual and automatic master data sets.

Automatic	Manual
Contains one item, the key item	Contains the key item and may contain related, non-key items
Updating related detail set updates master automatically	Must be updated manually before detail data sets
Contains current key values	Contains all valid key values
Use when manual update is unreasonable and data value checking is unnecessary	Use to check for valid key item values and to reduce data repetition by storing details with the key
Must be linked to at least one detail data set	May stand alone

Diagramming a Data Base

To show the relationship between the various sets in a data base, a data base block diagram is used. Each type of data set is represented by a different shape. Each data path is represented by a line. Here are the three data set shapes –

Manual Master **Automatic Master** **Detail Set**



Storing Data Entries

Entries are stored into automatic and manual master data sets by a hashing process which determines the record number. (Hashing was discussed in the File Management section, earlier in this chapter.) Hashing allows entries to be retrieved quickly. Entries are stored into detail data sets in serial fashion. That is, a new entry is placed in the first empty record.

Accessing Data Entries

The System 45 uses two traditional methods for retrieving data: serial and random access. IMAGE/45 employs four methods for retrieving data entries, serial, directed, calculated and chained. Each method has different purposes and advantages. The four methods are discussed next.

Serial Access

Serial mode is used with either master or detail data sets. It is the most basic and is useful when all entries in a set are to be accessed. When accessing in serial mode, IMAGE/45 starts at the record after the current record (the most-recently accessed entry) and sequentially examines each record until the next non-empty one is found. This entry is the one that is accessed. Empty records are skipped automatically.

Example

The following example show entries in the LIBRARY master data set. Serial access can be used to find all of the librarians. The arrows to the left show the order in which entries are retrieved in serial mode.

Record Number	Key Item PLANT_NAME	LIBRARIAN	Other Data
1			
2	BOISE	SANDY BARLOW	
3			
4	CORVALLIS	BECKY ASHBY	
5			
6			
7	DCD	ANITA NELSON	
8	GSD	STACY LARSEN	
9			
10			
11	SAN DIEGO	JOHN BABCOCK	

Directed Access

Another method of selecting the master or detail data entry to access is to specify its record number. This is known as a directed read and is similar to random access. If the specified record is empty, an error condition occurs.

Directed access is useful when the program has determined, in some manner, which record to access. Directed access can also be used to read entries in reverse order along a data chain, which is impossible otherwise.

Example

The following example shows entries in the BOOK detail data set. The record number of the desired data entry is 5.

Record Number	TITLE	AUTHOR	Other Data
1			
2	PROGRAMMING PROVERBS	LEDGARD, HENRY	
3			
4			
→ 5	CHEMICAL PRINCIPLES	MASTERSON, WILLIAM	
6			
7			
8	COMPUTER ARCHITECTURE	FOSTER, CAXTON	
9			

Rewinding a Data Set

A directed read to record 0 has the effect of "rewinding" the data set, allowing a subsequent serial read to retrieve the first non-empty record.

Chained Access

Chained access is used primarily to retrieve the entries in a detail data set which have the same value for a key item and are chained together. The related entry in the master set keeps pointers to the beginning (head) and end of the chain and the number of entries in the chain. The chain head pointer is retrieved prior to chained access. The pointer to the end of the chain is maintained to make it easier for the system to add another entry to the chain. Chained access can find only the next entry, not the previous one.

Chained access can also be used with manual master data sets to retrieve entries which are synonyms. This is useful if you have a large number of synonyms and wish to determine which values cause them.

Example

The following example shows how chained access in the INVENTORY detail data set can be used to find all of the books the borrower with EMPLOYEE_NO 9122 has checked out.

Record Number	Forward Chain Pointer	Backward Chain Pointer	Key Item EMPLOYEE_NO	Other Items
1	4	0	9122	
2	6	0	2359	
3	0	0	4105	
4	8	1	9122	
5	10	0	2311	
6	0	2	2359	
7				
8	0	4	9122	
9				
10	0	5	2311	

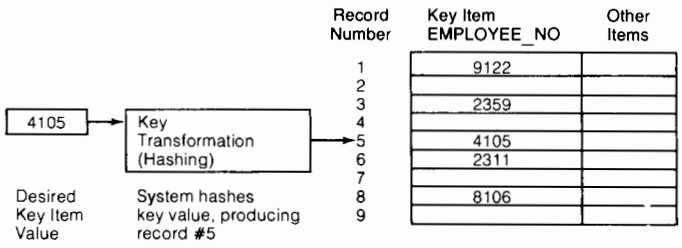
Head of Chain	End of Chain	Chain Count	Key Item EMPLOYEE_NO
1	8	3	9122

Calculated Access

Calculated access is used to access a master data set entry by specifying the key item value you want to access. This access method utilizes hashing, the process used when an entry is stored. Calculated access is an efficient method when you know the key item value you want.

Example

The following example shows retrieval of an entry using calculated access. Calculated access is used to retrieve information from the BORROWER manual master data set about a borrower with EMPLOYEE_NO 4105.



Data Base Files

There are three types of files associated with IMAGE/45 data bases.

Each IMAGE/45 data base has one **root file** which contains all the structural information about the data. This file must reside on a disc. The root file name is the same as the data base name declared in the Schema. The root file is created by the Schema Processor program. It can be copied to a disc or purged like any other file before the data base is actually created. A root file is designated by ROOT when a CAT operation is performed.

NOTE

The root file must always be in the same mass storage device while a data base is being accessed.

A **data set file** resides on a disc and contains all the entries for a particular data set. There is one file for each data set in the data base. Data set files are created and named by DBCREATE. A data set file is designated by DSET when a CAT operation is performed.

A **backup file** is created by the TBKUP utility program or DBBACKUP statement and contains a backup copy of all or a portion of the data base. There may be more than one backup file for a data base. The backup files are used to restore the data base after a system failure. A backup file is designated by BKUP when a CAT operation is performed.

Data Security

IMAGE/45 enables data to be protected from unauthorized access, since access to a data base requires a **password**. Each set has one or more passwords specified for it in the Schema; each of these passwords has either read-only or read/write access. In order to read from or write to a set, a valid password must have been used to open the data base. For example, a clerk may be given a password which allows him to have read but not write access to a set.

Data security can also be achieved by storing sensitive data sets on different discs and physically removing the discs from access by general users. This is explained further in Chapter 9.

Corrupt Data Base

A data base is said to be corrupt when either data or structural information has been lost. When this is the case, it may not be possible to access the data which is there. Thus, there are certain things which you **should not do** in order to avoid corrupting a data base. While a data base is open, **do not** –

Press the STOP key if not all data is written to the disc
Perform a reset operation (CONTROL-STOP)
Remove a disc from the drive
Fail to close the data base (discussed in Chapter 4)

In general, it is recommended that you let programs come to a natural halt and leave all discs in place if there is a chance that some data has not been written.

Data Backup

Data backup is an essential part of data base management. IMAGE/45 provides statements and a utility program for backing up your data base. If your data base is not backed up and some failure occurs, you face the possibility of losing all or part of your data.

Frequency of data backup depends on usage. Backup should be performed when significant changes are made to the data base.

Multiple-Volume Data Bases

IMAGE/45 enables creation of the root file and data set files of a data base on more than one disc medium, or **volume**. Each file must be on only one disc. A multiple-volume data base may be necessary if the data sets are large, or be desirable if some sets contain sensitive data that should not be generally accessible. The sets may reside on up to 24 different discs, with up to four **on-line** at one time. On-line volumes are specified with a VOLUME DEVICES ARE statement which specifies disc drives; the volumes in those drives are on-line.

Volumes

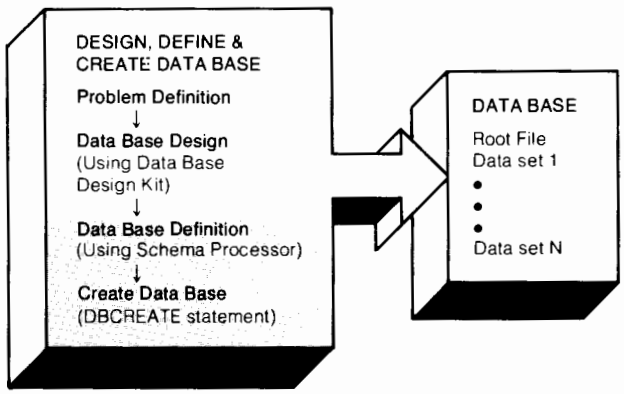
Multiple-disc data bases are made possible by the capability to give each disc a name or **volume label**, then reference the disc by that label, rather than by the mass storage unit specifier. When defining a data base, you can specify a volume label for each set. If no volume label is specified for a set, it resides on the same disc as the root file. Up to 23 volume labels can be specified. The root file disc is not referenced by a label, even if it has one, so sets on the root file disc are not given a volume label. During data base access, you specify which volumes are on-line, or currently accessible. Refer to Chapter 9, Advanced Data Base Concepts, for more information.

Chapter 3

Creating the Schema and Data Base

Introduction

After you have designed a data base, you must describe its structural definition to IMAGE/45. The design process is covered in the Data Base Design Kit; refer to the Design Kit if you haven't designed your data base. The **schema** is a block of text which defines passwords, items, sets, relationships, etc. After the schema text is entered and SAVEd in a data file, it is used by the Schema Processor program to create the root file. The root file is the directory to the data base. Then the data set files can be created with DBCREATE and accessed.



After the logical design of the data base is completed, there are a few decisions to make before entering the schema text. They are –

- Data base name
- Passwords and what type of access (read-only or read/write) each password has to each set.
- Name, type (manual master, automatic master or detail) and capacity of each data set.
- Name and type (long, short, integer or string) of each data item.
- The volume label if the set is to reside on a disc other than the one the root file is on.

Creating a Schema

The schema text is entered using the EDITLINE mode to enter a program of comments (lines beginning with !). Each line of the schema uses one program line (one comment) and can be followed by a schema comment denoted by <<comment>>. There can be up to 80 characters (schema line and comment) following the exclamation point. Any more are truncated. After the schema lines are typed, the program should be SAVEd. The Schema Processor program uses this data file to create the root file.

Schema Structure

A schema has three main parts: the password part, the item part and the set part. Additional statements are used to specify the data base name, to select Schema Instructions, and to designate the end of the schema. The password part specifies the passwords and associated password numbers. The item part specifies the type of each data item. The set part specifies the type of each data set, its capacity, passwords, paths, any volume, and which data items it contains.

The basic structure of a data base schema is –

```
BEGIN DATA BASE        data base name;

    PASSWORDS:
    [password number    password;]
     •
     •
     •
    [password number    password;]

    ITEMS:
     item name, [dimension] specifier [ <control number> ] ;
     •
     •
     •
     [item name, [dimension] specifier [ <control number> ];]

    SETS:
      { NAME]:      set name, M[ANUAL] [<[read list]/read/write list>] [, volume label];
      { ENTRY]:     key item name <path count>,[1]
for each                [item name, ]
manual                   •
master                   •
set                      •
                        [item name];
      { CAPACITY]:  maximum number of entries;

for each
automatic
master
set
      { N[AME]:     set name, A[UTOMATIC] [<[read list]/read/write list>] [, volume label];
      { E[NTRY]:    key item name <path count>;
      { C[APACITY]: maximum number of entries;

      { N[AME]:     set name, D[ETAIL] [<[read list]/read/write list>] [, volume label];
      { E[NTRY]:    item name [ <master set name> ],[1]
for each                [item name [ <master set name> ], ]
detail                   •
data                     •
set                      •
                        [item name [ <master set name> ] ];
      { C[APACITY]: maximum number of entries;

END.
```

Schema Comments

Comments can appear at the end of each line and are denoted by < < comment > >. A comment cannot be nested inside another one.

[1] If there are no other items, the entry line ends with a semicolon instead of a comma.

Parameters

This section defines the schema parameters. In general, there are two rules that apply to the entire schema. First, the lines BEGIN DATA BASE, PASSWORDS, ITEMS, SETS and END must each be on a separate line as shown in the syntax. Secondly, blanks in names are taken out.

Parameter — **Explanation**

data base name — A 1 to 4 character string consisting of upper and lowercase letters[1], digits 0 through 9 and the underscore character (_), beginning with a letter. This name becomes the root file name.

PASSWORD Part

password number — A unique integer from 1 to 31 which is associated with the password that follows it. The password number is used as a simple substitute for the password for specifying access to sets. If no password follows a password number, that line of the schema is ignored.

password — A string of 1 to 8 characters used to gain access to the data base and data sets. It can contain any ASCII character with a value >32 and <128, excluding semicolons. If a password is associated with more than one password number, the lowest one is used. If no passwords are specified, any string can be used to gain access to the data base.

ITEM Part — This part lists all data items in the data base. There can be up to **255** unique item names. Each item is numbered implicitly based on its position in the item part.

Programming Hint

Alphabetizing the items in the item part makes it easier to find a specific item.

---

item name — A unique 1 to 15 character string consisting of upper and lowercase letters[1], digits and the underscore character (_). It must begin with a letter.

dimension — An integer >0 which specifies the **number of subitems in a compound item**. 1 is the default value if it is omitted, indicating a simple item. The maximum value depends on the set and the items in it. The sum of the space needed by all items and all paths must be ≤1024 bytes. Path requirements are found in the section labeled Restrictions.

specifier — A specifier declares the item type, string or numeric. A numeric specifier can be one of the following –

Specifier	Type	Range	Item[2] Length
L	long (12-digit) numeric	±9.99999999999E±99	8 bytes
S	short (6-digit) numeric	±9.99999E±63	4 bytes
I	integer	−32 768 to 32 767	2 bytes

A **string specifier** is –

X followed by an even integer specifying the maximum number of characters.

Its length is 1 byte per character. As with compound items, the maximum length of any simple or compound string type depends on the set and the items in it. The sum of the space needed by all items and all paths must be ≤1024 bytes. Path requirements are found in the section labeled Restrictions.

control number — Any item can be assigned any integer from 0 to 127. The control number for any item can be retrieved using DBINFO (mode 102) and used as a flag to a program. For example, the control number could represent the number of significant digits of the item and be used for formatting purposes.

1 Nationalized characters may also be included if you have a local language keyboard.

2 These lengths are for simple items. To find the length of a compound item, multiply the dimension value by the item length.

SET Part

This part describes all sets in the data base. There can be up to **32 sets**, each having a unique name. Each set is numbered implicitly based on its position in the set part. The first set is set 1, the second is set 2, etc. There can be a total of **127 items** in a set. All items in a set must be unique and must have been defined in the item part.

NOTE

All master sets to which a particular detail set is linked must appear **before** that detail set in the set part. Specifying all masters, then all details and alphabetizing them makes it easier to find a set in the schema text.

---

set name

A unique 1 to 15 character string consisting of upper and lowercase letters[1], digits and the underscore character (_). It must begin with a letter.

read list
read/write list

Each list consists of password numbers or 0, separated by commas. Together, the lists specify which passwords have read-only and which have read/write access to the set. If a list is omitted or contains 0, **any** string has access to that particular set[2].

volume label

A string of 1 to 8 ASCII characters specifying the volume label of the disc on which the data set is to reside. It **must** be omitted when the set is to reside on the same volume as the root file. (Refer to Chapter 9 for more details on multiple-volume data bases.) Each volume label that appears in the schema is given a number based on when it was declared. The first volume declared is volume 1, the second is volume 2, etc. The root file volume is volume 0.

For master sets:

key item name

Each master set ENTRY part begins with the name of its key item. A key item name has the same form as a regular data item. Key items can't be compound items.

path count

An integer between 0 and 16 which specifies the number of key items in detail data sets to which the master has paths. A manual master with a path count of 0 is known as a **stand-alone master** and is not associated with any detail sets. **Automatic masters must have a path count of at least 1; they can't stand alone.**

item name

Manual master data sets can contain other data items in addition to the key. Item names in the set part must be defined in the item part.

maximum number of entries

The capacity of any set can be an integer between **1 and 32 767**. However, this number is limited by the size of the set and number of paths. The maximum size of any set is 32 767 256-byte records (8 388 352 bytes). If the media record length (items plus paths) is >256 bytes, the maximum possible capacity is reduced accordingly.

Here are some rules to follow when choosing the capacity of a master data set, which can increase efficiency –

- For master sets, never choose a power of 2 greater than 10. Additionally, it is best to choose a prime number[3].
- Choose a capacity which is at least 25% greater than the number of data entries you intend to have. This helps ensure rapid searches by reducing synonyms.

For detail sets:

item name (master set name)

In a detail set, any item name followed by a master set name is a key item for the detail set and is linked via a **path** to that master set. The master set must be previously defined. Though the two items need not have the same name, they must have identical specifiers in the item part. That is, they must both be the same numeric type or both be strings of the same length. A detail set can have up to 16 paths linking it to master data sets. A detail data set with no paths is known as a **stand-alone detail**.

1 Nationalized characters may also be included if you have a local language keyboard.

2 With QUERY/45, however, a valid or blank password must be used to access the set.

3 Appendix C contains a prime number table to aid you in choosing capacities.

Example

The following is the schema text for the sample data base.

```
10    ! $CONTROL LIST,ROOT,TABLE
20    ! $TITLE  "NOP Company Library Data Base"
30    ! BEGIN DATA BASE        LIBR;  << sample data base - NOP library >>
40    !
50    ! PASSWORDS:
60    !                        5     LIBRMGR;
70    !                        10    ENGINEER;
80    !
90    ! ITEMS:
100   !                        AUTHOR,          X50;  << 50-character string >>
110   !                        BORROW_DATE,     S;    << short precision >>
120   !                        BORROWER_NAME,   X50;
130   !                        CALL_NUMBER,     L;    << real precision >>
140   !                        COPY_NUMBER,     X10;
150   !                        EMPLOYEE_NO,     I;    << integer >>
160   !                        EMPLOYEE_PHONE,  X14;
170   !                        LIBRARIAN,       X50;
180   !                        LOCATION,        I;
190   !                        PHONE_NUMBER,    X14;
200   !                        PLANT,           X10;
210   !                        PLANT_ADDRESS,   3X40;  <<<compound>>
220   !                        PLANT_NAME,      X10;
230   !                        PRICE,           S;
240   !                        PUBLISHED_DATE,  S;
250   !                        PUBLISHER,       X30;
260   !                        SUBJECT,         X40;
270   !                        TITLE,           X60;
280   !
290   ! SETS:
300   !         NAME:          AUTHOR,AUTOMATIC(/5);   << 5 has read/write >>
310   !         ENTRY:         AUTHOR(1);              << 1 path >>
320   !         CAPACITY:      89;                     << choose a prime >>
330   !
340   !         NAME:          CALL_NUMBER,AUTOMATIC(/5);
350   !         ENTRY:         CALL_NUMBER(2);
360   !         CAPACITY:      89;
370   !
380   !         NAME:          SUBJECT,A(/5);
390   !         ENTRY:         SUBJECT(1);
400   !         CAPACITY:      53;
410   !
420   !         NAME:          TITLE,A(/5);
430   !         ENTRY:         TITLE(1);
440   !         CAPACITY:      89;
450   !
460   !         NAME:          LIBRARY,MANUAL(10/5);  << CLERK can only read >>
470   !         ENTRY:         PLANT_NAME(1),
480   !                        PLANT_ADDRESS,
490   !                        LIBRARIAN,
500   !                        PHONE_NUMBER;
510   !         CAPACITY:      13;
520   !
530   !  $PAGE
540   !         NAME:          BORROWER,M(10/5);
550   !         ENTRY:         EMPLOYEE_NO(1),
560   !                        BORROWER_NAME,
570   !                        LOCATION,
580   !                        EMPLOYEE_PHONE;
590   !         CAPACITY:      79;
600   !
610   !         NAME:          BOOK,DETAIL(/5,10);
620   !         ENTRY:         TITLE(TITLE),   << key item; link to  TITLE >>
630   !                        CALL_NUMBER(CALL_NUMBER), << another key item >>
640   !                        AUTHOR(AUTHOR),
650   !                        SUBJECT(SUBJECT),
660   !                        PUBLISHED_DATE,            << not a key item >>
670   !                        PUBLISHER,
680   !                        PRICE;
690   !         CAPACITY:      89;
700   !
710   !         NAME:          INVENTORY,D(10/5);
720   !         ENTRY:         CALL_NUMBER(CALL_NUMBER),
730   !                        COPY_NUMBER,
740   !                        PLANT(LIBRARY),
750   !                        EMPLOYEE_NO(BORROWER),
760   !                        BORROW_DATE;
770   !         CAPACITY:      193;
780   !
790   ! END.                                        << must have an END. >>
```

Restrictions
The following table summarizes restrictions on the values of various schema parameters.

Parameter	Maximum Value
Total number of items	255
Total number of sets	32
Total number of items in a set	127
Total number of paths from a master set	16
Total number of paths to any detail set	16
Total number of volume labels	23
Maximum media record length (items + paths)	1024 bytes
Maximum set size	32 767 256-byte records (8 388 352 bytes)

To determine the media record length required by a set entry, the following table can be used.

Set Type	Media Record Length (bytes)
Master	Entry length + 6 bytes + 6 bytes per path
Detail	Entry length + 4 bytes per path
Stand-alone detail	Entry length + 4 bytes

The entry length is the combined lengths of all the items in the set.

Schema Instructions

The three Schema Instructions, $CONTROL, $PAGE, and $TITLE specify page formatting control and select options during processing of the schema. They can be entered at the same time as the rest of the schema. A schema instruction can appear on a separate line anywhere in the schema and can not contain a comment. If a parameter list is included, it must be separated from the instruction name by at least one blank. Parameters are separated by commas.

The $CONTROL Instruction

$CONTROL parameter list

The $CONTROL instruction selects Schema Processor options. The options are indicated by the parameters which can include –

LIST or NOLIST — LIST causes each line of the schema to be listed on the printer you specify during the configuration part of the Schema Processor program. NOLIST disables the LIST option; only lines with errors are listed, followed by the error message.

ROOT or NOROOT — ROOT causes the Schema Processor to create a root file if no errors were detected. NOROOT prevents the Schema Processor from creating a root file. You may want to specify NOROOT until you are satisfied with the schema; otherwise, you must PURGE the root file to change the schema.

TABLE or NOTABLE	TABLE causes the Schema Processor, if there are no preceding errors, to print a table of summary data set information following the listing. The information for each set includes – **Data set name** **Data set type**: M(anual), A(utomatic) or D(etail) **Field (Item) count** (number of items in the set) **Path count** (number of associated paths) **Entry length** (total number of bytes used by the items) **Media record length** (entry length + path requirements) **Data set capacity** (maximum number of entries) **Physical Records** (number of 256-byte physical records required by the data set) **Volume Name** (where data set is to reside; it is the root file volume if one isn't specified) NOTABLE suppresses the TABLE option.
ERRORS= value	Sets the maximum number of allowable errors. If this value is exceeded during processing, the Schema Processor is terminated. The range of the value is 0 through 999.
LINES= value	Sets the maximum number of lines to be printed on a page. The value can be an integer from 20 through 999.

The $CONTROL parameters can be in any order and must be separated by commas. The default values for the various options are –

LIST, ROOT, TABLE, ERRORS=100, LINES=66

The $PAGE Instruction

$PAGE ["character string"]

The $PAGE instruction causes the schema to continue listing on a new page (unless NOLIST has been specified in a previous $CONTROL command). The characters within the quotes and two blank lines are printed at the top of the page.

If a string is specified, it replaces the title string specified in any previous $PAGE or $TITLE instruction; otherwise, the title remains unchanged. The maximum length is 30 characters; any more characters than 30 are truncated. (A quote mark (") can be printed by placing double quotes ("") in the string.)

The $TITLE Instruction

$TITLE ["character string"]

The $TITLE instruction specifies a title to be printed at the top of each new page of the schema listing. The title replaces any title specified by a previous $TITLE or $PAGE. If the string is omitted, no title is printed until a subsequent $TITLE or $PAGE instruction specifies one.

A quote mark (") can be printed by placing double quote marks ("") in the string. The maximum length of the title is 30 characters; any more characters than 30 are truncated.

Processing the Schema

After you are satisfied that your schema is correct, it can be processed by loading and running the Schema Processor program supplied on the Utility tape. The Schema Processor reads and processes the SAVEd Schema text file according to the options specified by a $CONTROL instruction (or the default values if there is no $CONTROL), finding any syntax or logical errors. It prints (again, according to $CONTROL) a listing of the schema, any errors, and a summary table. If there are no errors, the data base root file is created.

To load and run the Schema Processor program, execute the following statement. The msus is necessary if the program is not on the current default mass storage device.

```
LOAD "SCHEMA [msus]",1
```

Necessary Files

A file, "Confg", is used by the Schema Processor. It must be on either the current default mass storage device or a mass storage device defined by VOLUME DEVICES ARE. Two other files are used during processing of the schema. "SCHERR" contains all the necessary error messages. "SCRH" is created by the Schema Processor as a scratch file, then is purged when schema processing is complete. It is 85 records long, so there must be enough room on the specified disc to contain it.

Information to Enter

First, you are asked to enter the name of the SAVEd Schema text file. The msus must also be entered if the file is not on the current default mass storage device. Next, you can change the configuration for the schema processing. By pressing the appropriate Special Function Key (as indicated on the CRT), you can change the location of –

- the error file
- the scratch file, "SCRH" (must be on a disc)
- the root file (must be on a disc)
- the printer for the schema listing, table, etc.

When the configuration is correct, press key 7.

Schema Processor Results

As it runs, the Schema Processor produces various output which includes –

- schema listing and summary table (according to $CONTROL)
- total number of 256-byte physical records needed for the data sets and root file
- total number of items and sets

Any time an error is encountered, the line with the error is printed (even if NOLIST is in effect), followed by an error message. Errors prevent creation of the root file and printing of some summary information.

If there are errors, you can GET the file containing the schema, edit it, RE-SAVE it, then LOAD and rerun the Schema Processor.

Creating the Root File

If there are no errors (and NOROOT is not specified by $CONTROL), the root file is created and a message printed to that effect. It is given the name of the data base that was specified in the schema. Errors in $CONTROL instructions do not prevent creation of the root file. Once the root file is created, the data base can be created.

Programming Hint

You may want to specify NOROOT in $CONTROL until you are satisfied with the schema. Otherwise, you must PURGE the root file if you wish to change the schema.

Unreferenced Data Items

If there are any data items specified in the item part which are not used in any sets, the message UNREFERENCED DATA ITEMS: is printed, followed by the names of the unreferenced items. Their existence is not considered an error. However, you should check to make sure they were not left out of a set. Otherwise, remove them from the item part; they each use 20 bytes in the root file.

Example

Here is the output when the Schema Processor is run using the file containing the schema for the sample data base.

```
PAGE 1     Hewlett-Packard  9845B   ---   Schema Processor  Rev. A


 $CONTROL LIST,ROOT,TABLE
 $TITLE  "NOP Company Library Data Base"
 BEGIN DATA BASE        LIBR;  << sample data base - NOP library >>

 PASSWORDS:
                        5     LIBRMGR;
                        10    ENGINEER;

 ITEMS:
                        AUTHOR,           X50;  << 50-character string >>
                        BORROW_DATE,      S;    << short precision >>
                        BORROWER_NAME,    X50;
                        CALL_NUMBER,      L;    << real precision >>
                        COPY_NUMBER,      X10;
                        EMPLOYEE_NO,      I;    << integer >>
                        EMPLOYEE_PHONE,   X14;
                        LIBRARIAN,        X50;
                        LOCATION,         I;
                        PHONE_NUMBER,     X14;
                        PLANT,            X10;
                        PLANT_ADDRESS,    3X40;  <<compound>>
                        PLANT_NAME,       X10;
                        PRICE,            S;
                        PUBLISHED_DATE,   S;
                        PUBLISHER,        X30;
                        SUBJECT,          X40;
                        TITLE,            X60;
```

```
SETS:
        NAME:           AUTHOR,AUTOMATIC(/5);   << 5 has read/write >>
        ENTRY:          AUTHOR(1);              << 1 path >>
        CAPACITY:       89;                     << choose a prime >>

        NAME:           CALL_NUMBER,AUTOMATIC(/5);
        ENTRY:          CALL_NUMBER(2);
        CAPACITY:       89;

        NAME:           SUBJECT,A(/5);
        ENTRY:          SUBJECT(1);
        CAPACITY:       53;

        NAME:           TITLE,A(/5);
        ENTRY:          TITLE(1);
        CAPACITY:       89;

        NAME:           LIBRARY,MANUAL(10/5);  << CLERK can only read >>
        ENTRY:          PLANT_NAME(1),
                        PLANT_ADDRESS,
                        LIBRARIAN,
                        PHONE_NUMBER;
        CAPACITY:       13;
```

```
PAGE  2           LIBR              NOP Company Library Data Base

        NAME:           BORROWER,M(10/5);
        ENTRY:          EMPLOYEE_NO(1),
                        BORROWER_NAME,
                        LOCATION,
                        EMPLOYEE_PHONE;
        CAPACITY:       79;

        NAME:           BOOK,DETAIL(/5,10);
        ENTRY:          TITLE(TITLE),    << key item; link to  TITLE >>
                        CALL_NUMBER(CALL_NUMBER), << another key item >>
                        AUTHOR(AUTHOR),
                        SUBJECT(SUBJECT),
                        PUBLISHED_DATE,            << not a key item >>
                        PUBLISHER,
                        PRICE;
        CAPACITY:       89;

        NAME:           INVENTORY,D(10/5);
        ENTRY:          CALL_NUMBER(CALL_NUMBER),
                        COPY_NUMBER,
                        PLANT(LIBRARY),
                        EMPLOYEE_NO(BORROWER),
                        BORROW_DATE;
        CAPACITY:       193;

END.                                        << must have an END. >>
```

```
DATA SET          TYPE  FLD  PATH  ENTR  MED  CAPACITY  PHYSICAL VOLUME
NAME                    CNT  CNT   LGTH  REC            RECORDS
--------          ----  ---  ----  ----  ---  --------  -------  ------
AUTHOR             A     1     1    50    62     89        22
CALL_NUMBER        A     1     2    8     26     89        10
SUBJECT            A     1     1    40    52     53        11
TITLE              A     1     1    60    72     89        26
LIBRARY            M     4     1    194   206    13        11
BORROWER           M     4     1    68    80     79        25
BOOK               D     7     4    196   212    89        74
INVENTORY          D     5     3    34    46     193       35
                                          ROOT FILE LENGTH:  7
                                TOTAL SECTORS INCLUDING ROOT:  221

NUMBER OF ERROR MESSAGES:   0
DATA ITEM COUNT:   18    DATA SET COUNT:   8

ROOT FILE LIBR GENERATED
```

Schema Error Messages

Whenever the Schema Processor encounters a syntax or logical error in a schema line, the line is listed, followed by an error message. It then continues processing. An exception to this is if PASSWORDS: is omitted or is not on a separate line. In this case, processing stops.

If a terminator such as a semicolon, or a delimiter such as a comma is missing, the Schema Processor may ignore an entire line, causing subsequent errors.

Any error other than an error in a schema instruction or an unreferenced data item prevents creation of the root file. A complete list of Schema Processor errors is given in the back of this manual. Each unreferenced data item causes the root file to be 20 bytes longer than is necessary.

File Errors

An error involving the error file, scratch file or the data file containing the schema text causes the Schema Processor to return to the Configuration Section and a message to be printed.

Creating the Data Set Files – DBCREATE

After the root file is created, the final step in preparing for data base operations is to create and name the data set (DSET) files so that information can be put into them. This is done with the DBCREATE statement, which is analogous to the CREATE statement for data files.

DBCREATE data base name [; maintenance word][, set list / * / msus / volume specifier]

Parameters

data base name — A string variable containing the data base name as declared in the Schema. The msus of the root file should follow the name as part of the string if it is not on the current default mass storage device.

maintenance word — A string expression (of which only the first 6 characters are used) which can be specified during the first DBCREATE for a data base. It is used as a protect code that must be used with any subsequent DBCREATE, DBERASE, DBPURGE, DBBACKUP, DBLOAD and DBUNLD. The maintenance word can only be changed if the data base is purged with DBPURGE and a new DBCREATE is performed.

The final parameter has four possible variations; it is used primarily in the creation of multiple-volume data bases. It tells the system to create some of the sets, either by specifying which sets to create (first two parameters) or that a particular medium is there and to create the sets designated in the schema to reside on it (second two parameters). If one of the four final parameters is not included, DBCREATE creates all the sets in the data base.

Set volumes are specified in the schema. If no volume is specified for a set, it is created on the same disc as the root file. When creating a multiple-volume data base, it is essential that each set be created on the volume which was specified in the schema. For more information and an example of a multiple-volume DBCREATE, refer to Chapter 9, Advanced Data Base Concepts.

set list — A string expression which contains set numbers separated by commas. It lets you create only selected sets.

* — A string expression which contains an asterisk ("*"); it tells the system to create all data sets not already created.

msus — a string expression of the form –
; device type [select code [, controller address | 9885 unit code [, unit code]]]
It tells the system to create the sets for that medium.

volume specifier — a string expression containing the volume label preceded by a comma (",volume label"). If you specify a volume, it must be on-line. This is done with a VOLUME DEVICES ARE statement. It tells the system to create the sets for that medium.

Possible Errors

The following errors are possible during a DBCREATE.

Value	Meaning
208	Needed volume not on-line
209	Operation not allowed on tape
212	Set number in the set list cannot be larger than the set count
218	Volume name is not part of the data base
220	Improper or illegal maintenance word
225	Improper utility version number in root file
226	Corrupt data base; must purge and redefine. Purge the root file and run the Schema Processor
229	Operation not allowed when the data base is open
230	Improper set list or duplicate set numbers in the set list

Example

The following example program creates the data set files for the sample data base.

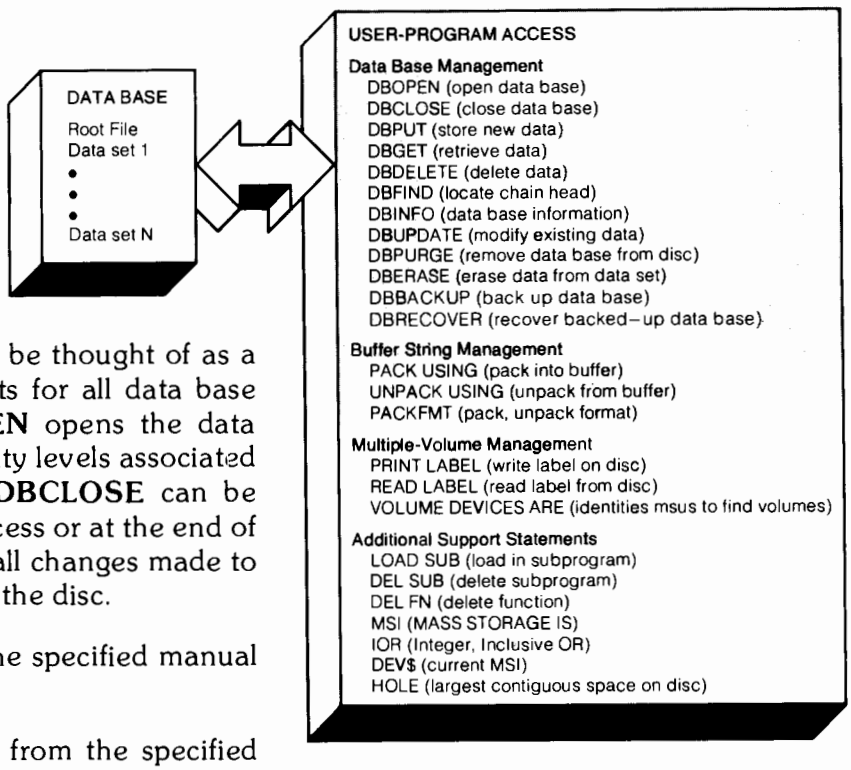
```
10    !  This program creates the sample data base
20    !
30    DIM B$[6]
40    B$="LIBR"
50    MASS STORAGE IS ":F8"      ! Root file is on :F8
60    DBCREATE B$;"BOOKS"        ! Create entire data base
                                   BOOKS is maintenance word
70    END
```

For a multiple-volume example, see Chapter 9.

Chapter 4
Data Base Access

Introduction

After your data base is designed and the root file and data sets are created, you are ready to begin accessing your data base. There are eleven statements which provide flexibility in accessing data in a data base.



DBOPEN/DBCLOSE can be thought of as a necessary pair of statements for all data base access operations. **DBOPEN** opens the data base according to the security levels associated with the password used. **DBCLOSE** can be used after any data base access or at the end of the program to insure that all changes made to the data base are written to the disc.

DBPUT adds an entry to the specified manual master or detail data set.

DBGET retrieves an entry from the specified master or detail set.

DBFIND locates the head of detail data chain so that a chained-access DBGET can be performed.

DBUPDATE updates an entry in the specified set.

DBDELETE deletes an entry from the specified set.

DBINFO provides information about data items, data sets, data paths and volumes.

PACK/UNPACK/PACKFMT provide buffer management for the data base buffer string; the data base buffer string is used as an intermediate location for data as it is passed between the data base and the program. **PACKFMT** specifies the variables and their formats; **PACK** and **UNPACK** put data into and take data from the buffer string on its way to and from the data base.

NOTE
In order to perform any data base access operations, the root file must be mounted in an accessible disc drive at all times.

Considerations

The mainframe statements LOADALL and STOREALL should **not** be used within any IMAGE/45 program. Otherwise, an error may occur.

Status Array

Each of the data base manipulation statements checks various conditions and places information concerning their status into an integer array, called the **status array**. It is essential to check certain elements of the array after each data base manipulation statement is executed. The meanings of particular elements and possible values are discussed with each statement. After an error, the status array contains different information. This information is covered in the Error Messages at the end of this manual.

Therefore, before performing data base operations, your program should dimension an array to be used as the status array. It should be integer and have ≥ 10 elements[1]. Throughout this manual, "Status" is used as the name of this array. The following can be used to dimension it –

```
10 INTEGER Status(1:10)
```

Condition Word

The first element in the status array is known as the **condition word**. After a successful data base operation, its value is 0. If there is an error, its value is non-zero. The condition word should be checked after each data base operation; if it is non-zero, an error has occurred and the operation was not performed. Steps should be taken to correct the error. If it is necessary to stop the program, make sure the data base is closed.

Current Record

Any time you read or write a data entry with DBGET or DBPUT, the entry involved becomes the **current record** for that set. When a data base is opened, the current record pointers for each set are set to record 0. The current record is the one used by DBUPDATE and DBDELETE.

Data Base Parameters

All of the data base access statements are composed of various parameters. In general, there are two concepts to keep in mind when entering one of these statements.

- No data base parameter may be a reference to a multiple-line function (FN).
- In string parameters, make sure that there are no extra blanks within the value. For example, if you specify "SET 1" when the name of the set is actually "SET1", IMAGE/45 uses only the characters before the blank and interprets this at "SET".

1 If the status array has more than ten elements, the first ten are used.

Data Base Guidelines

When writing and using IMAGE/45 data base programs, there are certain guidelines to keep in mind. These guidelines pertain to various topics and are covered in the appropriate places in this manual. For your convenience, they are summarized inside the back cover of this manual.

Opening the Data Base – DBOPEN

Anytime you wish to use a data base, the DBOPEN statement must be used to initiate access to it. Up to five data bases can be open at any time. The syntax of DBOPEN is –

```
DBOPEN (data base name, password, mode, status array)
```

Parameters

data base name	A string variable which contains the data base name **preceded by two blanks**. The msus of the root file should follow the name as part of the string if it is not on the current default mass storage device. The system fills the first two positions with digits, so this variable should not be altered while the data base is open. Otherwise, the data base may be corrupted.
password	A string expression containing a password. Passwords are defined in the schema.
mode	A numeric expression equal to 3, 8 or 11. If the value is not an integer, it is rounded.
status array	The array identifier for the status array (the array name followed by (*), as in Status (*)).

DBOPEN Modes

There are three DBOPEN modes, which grant read-only or read/write access.

Mode 3: **read/write access with immediate posting.** Allows both read and write access to the data base. Each change to the data base (by DBPUT, DBUPDATE, DBDELETE) is written immediately to the disk. **This mode should be used unless speed is critical, since it provides the highest level of data integrity.**

Mode 8: **read-only access.** Allows only read operations to be performed. The data base can't be altered by adding, updating or deleting entries. A data base may be opened more than once in mode 8. This is discussed later in this section.

Mode 11: **read/write access with buffering.** Allows both read and write access to the data base. Changes in the data base are kept in a memory buffer until the buffer is full or a DBCLOSE mode 4 is performed. This mode minimizes the number of disc accesses and is the mode to use when speed is critical. However, since data is kept in memory, there is a greater chance of corrupting the data base. For this reason, the STOP key should **not** be pressed if there is any chance that there is data still in the buffer. Frequent use of DBCLOSE mode 4, which writes buffered data to the disc, is recommended.

Status Array

The status array contains the following information after a successful DBOPEN –

Element	Value
1	0 (Condition word – error if non-zero)
2	Password number of password used
3	Number of bytes of memory used for global data base block[1] (0 is returned if it is not the initial open of the data base)
4	Number of bytes of memory used for local data base block[1]
5	0
6	401 (identification number for a DBOPEN statement)
7	line number of DBOPEN statement
8	number of changes in the data base since the last complete backup, with a maximum of 2047. This applies only to the initial DBOPEN for a data base; 0 is returned if the data base is opened another time
9	mode number of DBOPEN
10	integer – for system use only

The following values for the Condition Word are possible after an unsuccessful DBOPEN.

Value	Meaning
−1	Improper data base name; already have read/write access to the data base
−10	The maximum number of data bases (5) already opened
−11	First two blank characters missing from data base name parameter
−21	Invalid password
−31	Invalid mode value
−74	Root file name has been changed; this is not allowed
−91	Not a valid root file
−92	Data base requires creation
−94	Corrupt data base; may be opened in mode 8 only
−95	DBOPEN not allowed while a DBBACKUP or DBRECOVER is in progress
94	Corrupt data base has been successfully opened in mode 8
5xx	Volume is on-line, but created data set xx is missing

Possible Errors

The following errors are possible during a DBOPEN. In addition, mass storage errors are also possible.

Value	Meaning
209	Operation not allowed on tape
210	Bad status array
219	Out of user read/write memory. If the data base was opened in a subprogram, try opening it in the main program instead.

1 This memory is taken from user read/write memory when a data base is opened and is deallocated when the data base is closed. Refer to Chapter 9 for additional information about data base memory management.

Example

Here is an example which opens the sample data base for read/write access –

```
10    !  This program segment illustrates DBOPEN  ************
20    !
30    DIM Base$[6],Password$[8]
40    INTEGER Status(1:10)        ! Status array
50    Base$="  LIBR"              ! Root file name with 2 LEADING BLANKS
60    Password$="LIBRMGR"         ! Password
70    MASS STORAGE IS ":F8"       ! Location of root file
80    !
90    DBOPEN (Base$,Password$,3,Status(*))
100   IF Status(1)<>0 THEN Dberr ! ALWAYS CHECK STATUS ARRAY;
                                   branch to error routine if needed
110   !
120   !
130   ! REST OF PROGRAM FOLLOWS     ***********************
140   !
150   STOP
160 Dberr:                             ! Error routine if Status(1)<>0
170   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
180   DBCLOSE (Base$,X,1,Status(*))    ! Close if error occurs
190   END
```

Multiple DBOPENs

A data base can be opened up to five times in read-only mode, mode 8, using a different string variable containing the data base name each time. The system differentiates each occurrence of a DBOPEN by placing different values in those first two characters; this is why there must be two blanks before the name. However, a data base opened with mode 3 or 11 can be opened only once.

After each DBOPEN on a data base, every data set is given a pointer to indicate the current record. This allows there to be more than one current record per set. One use for multiple opens on the same data base is to restrict access to certain sets by certain users, since each DBOPEN requires that a password be given.

A maximum of five DBOPENs can be current at one time. That is, if five DBOPEN statements have been executed, a DBCLOSE mode 1 must be executed before another DBOPEN can be performed. Thus, if more than one data base has been opened, no data base can be opened five times.

Closing the Data Base – DBCLOSE

The DBCLOSE statement has two purposes. One mode is used when you have completed all the data base operations you currently want to do. The DBCLOSE statement is used to close the data base in an orderly fashion. **This must be done to avoid corrupting the data base.** Another mode of DBCLOSE is used to write all buffered information to the disc. This mode is used primarily when mode 11 was used to open data base.

DBCLOSE (data base name, variable, mode, status array)

Parameters

data base name	The same string variable containing the data base name used by DBOPEN.
variable	Any numeric or string expression. It is ignored, but included for compatibility with other IMAGE systems.
mode	A numeric expression equal to 1 or 4.
status array	The array identifier for the status array (the array name followed by (*)).

DBCLOSE Modes

The mode parameter describes the type of close.

Mode 1: Simple Close: The data base is closed and all changes are written to the disc. This sets the first two characters of the data base name variable back to blanks and frees memory used by DBOPEN.

Mode 4: Write buffered information to the disc. This mode updates data and structural information, but **leaves the data base open**. It is essential that DBCLOSE mode 4 be used often after performing DBPUTs, DBUPDATEs and DBDELETEs to ensure data integrity. This is important when DBOPEN mode 11 was used.

DBCLOSE Mode 4 is especially useful with multiple-volume data bases. It is essential to use it before removing a volume from a drive to ensure that all data has been written to it.

NOTE

DBCLOSE is an **essential** operation, whether it be periodic mode 4 closes or a mode 1 close at the end of the program.

Status Array

The status array contains the following information after a successful DBCLOSE –

Element	Value
1	0 (Condition Word – error if non-zero)
2	} unchanged
3	
4	
5	0
6	403 (identification number for a DBCLOSE statement)
7	line number of DBCLOSE statement
8	0
9	mode number of DBCLOSE
10	integer – for system use only

The following values for the Condition Word are possible after an unsuccessful DBCLOSE.

Value	Meaning
–11	Improper data base specified or data base not open. The data base name should not have been changed since the DBOPEN
–31	Invalid mode value
–74	Root file name has been changed; this is not allowed

Possible Errors

The following errors are possible during a DBCLOSE. In addition, mass storage errors (with numbers in the 70's and 80's) are possible.

Value	Meaning
210	Bad status array

Closing the Data Base Following an Error

Closing the data base is an essential step. Should an error occur in the program, the program may stop with the data base still open. Another area of concern is an I/O error which can only be recovered from by pressing the STOP key. In both cases, it is essential that the data base be closed. This can be done by executing the DBCLOSE statement from the keyboard.

Example

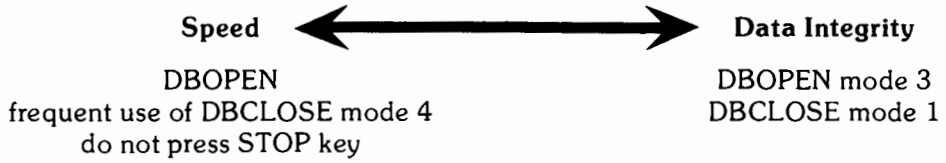
This program section closes the sample data base when you are done using it.

```
10    !  This program illustrates DBOPEN and DBCLOSE Mode 1
20    !  DBCLOSE mode 4 is used to write buffered information to disc
30    !
40    DIM Base$[9],Password$[8]
50    INTEGER Status(1:10)
60    Base$="  LIBR:F8"           ! Root file name with 2 LEADING BLANKS
                                  Root file location follows name
70    Password$="LIBRMGR"         ! Password
80    !
90    DBOPEN (Base$,Password$,3,Status(*))
100   IF Status(1)<>0 THEN Dberr       ! Branch to error routine
110   !
120   ! ************ REST OF PROGRAM HERE ******************
130   !
140   DBCLOSE (Base$,X,1,Status(*))    ! Close data base before end of program
150   IF Status(1)<>0 THEN Dberr
160   !
170   STOP
180 Dberr:                        ! Error routine if Status(1)<>0
190   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
200   DBCLOSE (Base$,X,1,Status(*))
210   END
```

Speed/Data Integrity Tradeoffs

In any data base operations, there is a tradeoff between speed and data integrity. Any time there is new data in memory buffers which hasn't been written to a disc, there is a chance of data loss. However, your application may require fast acquisition and manipulation of large amounts of data, which means the time needed to write to a disc can't be taken immediately.

There are two DBOPEN read/write modes to give you a choice. Maximum speed can be obtained by using DBOPEN mode 11 which buffers the data. DBCLOSE mode 4 should be used as often as possible to write data to the disc. Maximum data integrity can be obtained using DBOPEN mode 3 which writes any changes to the disc immediately. This mode is recommended unless speed is critical.



Using CHECKREAD

The mainframe statement CHECKREAD can be used to provide added integrity in data base operations. It must be in the program before the DBOPEN or it has no effect on data base operations. If a CHECKREAD is specified before a DBOPEN, it remains in effect until the data base is closed; if CHECKREAD OFF is later specified while the data base is open, it is ignored.

Using CHECKREAD in this manner may make data base operations slower and uses more memory. It is recommended that it be used only with flexible disks.

The Data Base Buffer String

Data is passed to and from the data base one entry at a time. An entry consists of values for all the items in a data set. Any time data is passed between a program and a data base, it is passed in a simple string, which serves as a buffer or intermediate location between the data base and the program variables. This **data base buffer string** contains values for the items in the data set. This data is condensed, normally in accordance with the data item formats specified in the schema. There may be more than one string variable serving as a buffer string; you may want to have one for each set.

Three statements are used for managing the data base buffer string. **PACKFMT** is used to specify the format of how the data is placed into the buffer string. This format is specified by the variables you want to pack and unpack. **PACK** places values of program variables into the buffer string in condensed form prior to entering the data into the data base with DBPUT or DBUPDATE. **UNPACK** is used after data is retrieved from the data base and put into the buffer string with DBGET or DBINFO. It takes the condensed values in the buffer string and puts them into program variables.

PACK and PACKFMT are similar to the PRINT USING and IMAGE statements. Additionally, the three buffer management statements can be used with READ# and PRINT# to compact data which is stored in a data file.

Buffer String Size

The buffer string, or a substring, if used, must be dimensioned large enough to contain an entire data set entry. The longest possible entry is 1022 bytes (characters). If the string is too short, error 236 occurs. The necessary length can be determined by the summary table which is printed for each set by the Schema Processor. 'Entry Length' (ENTR LGTH) indicates the length of each set in bytes.

The PACKFMT Statement

The PACKFMT (pack format) statement specifies a list of variables which determines the format to be used by PACK and UNPACK as the source or destination for the buffer string. There can be many PACKFMT statements in a program. The data set being accessed determines which format is needed by PACK and UNPACK.

PACKFMT pack list

The pack list can contain numeric variables, string variables, substrings, array identifiers and skip fields separated by commas. The format should correspond to the items in the data set definition in the schema, in order and in size.

NOTE

Use care when developing the pack lists for PACKFMT statements. A discrepancy in the formats used for storing and retrieving a data entry may cause unexpected results, including errors in PACK or UNPACK statements.

If the format you are defining is too long to fit in one PACKFMT statement, multiple PACKFMT statements can be used with multiple PACK or UNPACK statements. Specifically, you can PACK using the first PACKFMT into the first part of the buffer string by using a substring. Then, you can PACK using the second PACKFMT into the second part of the buffer string using a substring. This can be done as many times as needed. The same method can be used to UNPACK a buffer string with a long format.

Packing and Unpacking Strings

If a PACKFMT statement would contain only strings, it need not be PACKed or UNPACKed. The strings can be concatenated together and put directly into the data base buffer string rather than being packed. After retrieving data from the data base, substrings can be used rather than UNPACK.

Skip Fields

A **skip field** is used to ignore character positions in the buffer string. It is specified in the pack list with an X which is preceded by an integer to indicate how many characters to skip. 1X skips 1 byte, 2X skips 2 bytes, etc. This length should be the length of one or more items which you want to skip.

Using skip fields makes the PACK and UNPACK operations faster. A skip field in the PACKFMT for a PACK operation (prior to a DBPUT or DBUPDATE) can be used when a value currently in the buffer string from a previous PACK or UNPACK is the value that you want to put in the data base. Use a skip field when you are adding several entries to a set and those entries have the same value for a particular data item. Another use is to skip the items you do not want altered by a DBUPDATE.

A skip field in the PACKFMT for an UNPACK operation (following a DBGET or DBINFO) can be when you only want to retrieve values for certain items in the set and do not care about the others.

NOTE

If you fill the buffer string with data, later give it the value of the null string (no characters), then perform a PACK or UNPACK having skip fields in the format, those skipped characters contain the information that was there before the string was given the null value.

Item Lengths

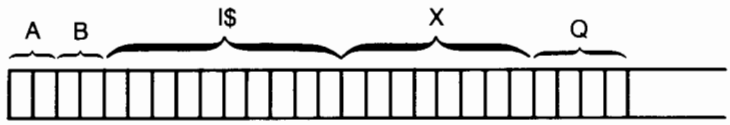
To determine how many bytes are needed to determine the buffer string length or to skip an item, refer to the following table.

Variable Type	Number of Bytes
Long Real	8
Short Real	4
Integer	2
String	1 per character in the dimensioned (or substring) length. If the current length is less, the string is padded with blanks.

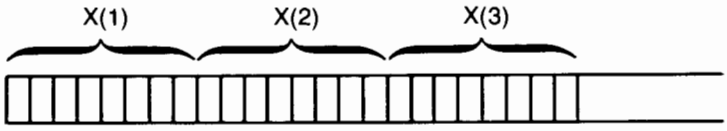
Examples

Here are two examples illustrating how data is represented in the buffer string.

```
 10  INTEGER A,B
 20  DIM I$[10]
 30  REAL X
 40  SHORT Q
  .
  .
  .
100  PACKFMT A,B,I$,X,Q
```




```
 5   OPTION BASE 1
 10  DIM X(3)
  :
 50  PACKFMT X(*)
```

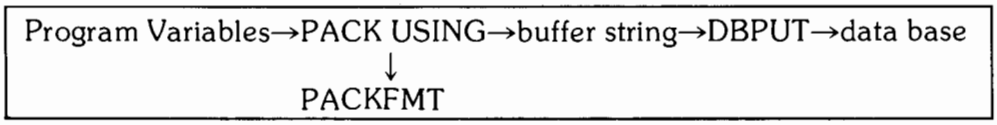


The PACK Statement

The PACK statement transfers values of program variables into the buffer string, according to the specified format. It is used before entering data into the data base with DBPUT and DBUPDATE.

PACK USING line identifier; buffer string

The following diagram illustrates how values of program variables are put into a data base.



Parameters

line identifier	The line identifier (line number or label) of the desired PACKFMT statement.
buffer string	The string variable or substring which serves as a data base buffer string. It must be long enough to contain all of the items in the PACKFMT statement. If the pack list in the PACKFMT statement is shorter than the dimensioned (or substring) length of the buffer string, the rest of the buffer string is filled with blanks.

Considerations

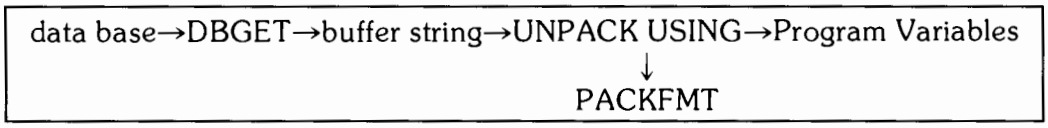
When strings specified in the PACKFMT are packed, they are padded, if necessary, with blanks up to their dimensioned (or substring) length.

The UNPACK Statement

The UNPACK statement transfers data from the buffer string to variables in a pack list. It is used after retrieving data from a data base with DBGET or DBINFO.

UNPACK USING line identifier; buffer string

The following diagram illustrates how values from a data base are put into program variables.



Parameters

line identifier	The line identifier (line number or label) of the desired PACKFMT statement.
buffer string	The string expression which serves as a data base buffer string. Its current length (number of characters it currently contains) must be long enough to contain all of the items in the pack list of the PACKFMT statement. Otherwise, an error occurs.

Considerations

When strings specified in the PACKFMT are unpacked, they are padded with blanks if their current length is less than the dimensioned (or substring) length of the string in the PACKFMT.

PACK/UNPACK Errors

The following errors are possible during an unsuccessful PACK or UNPACK.

Value	Meaning
17	Subscript out of range
18	Substring out of range or string too long
234	Referenced line not a PACKFMT statement
236	Destination string length too short in PACK; or source string current length too short in UNPACK
237	List item length greater than 32 767 in PACK or UNPACK
238	Numeric conversion error in UNPACK; improper short or real number
239	UNPACK requires a source string of greater length

Example

The following program illustrates PACKFMT, PACK and UNPACK.

```
10    !  This program illustrates PACKFMT, PACK USING and UNPACK USING
20    !
30    DIM Base$[9],Password$[8],Buf$[200]
40    INTEGER Status(1:10)
50    Base$="  LIBR:F8"
60    Password$="LIBRMGR"
70    DBOPEN (Base$,Password$,3,Status(*))
80    IF Status(1)<>0 THEN Dberr       ! Branch to error routine
90    !
100   ! ************ DIMENSION AND ASSIGN VARIABLES FOR 'BOOK' DETAIL SET:
110   !
120   DIM Title$[60],Author$[50],Subject$[40],Publisher$[30]
130   REAL Call_number
140   SHORT Published_date,Price
150   Title$="PASCAL: USER MANUAL AND REPORT"
160   Call_number=7516462
170   Author$="WIRTH,NIKLAUS"
180   Subject$="COMPUTER PROGRAMMING"
190   Published_date=FNEncode_date(1,1,1975)    ! SEE SUBPROGRAM BELOW *******
200   Publisher$="SPRINGER-VERLAG"
210   Price=6.95
220   !
230   ! ****************** DECLARE THE FORMAT FOR THE 'BOOK' DETAIL SET:
240   !
250 Book_fmt: PACKFMT Title$,Call_number,Author$,Subject$,Published_date,Publish
er$,Price
260   !
270   ! ******************************  NOW PACK THE BUFFER STRING:
280   !
290   PACK USING Book_fmt;Buf$
300   !
310   ! ********DATA CAN NOW BE WRITTEN TO THE DATA BASE.
320   ! ******* AFTER READING FROM A DATA SET, UNPACK THE
                BUFFER  STRING TO PUT VALUES INTO PROGRAM VARIABLES:
330   UNPACK USING Book_fmt;Buf$
340   !
350   !
360   DBCLOSE (Base$,X,1,Status(*))
370   IF Status(1)<>0 THEN Dberr
380   !
390   STOP
400 Dberr:                          ! Error routine if Status(1)<>0
410   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
420   DBCLOSE (Base$,X,1,Status(*))
430   END
440   !
450   ! ********************************************************************
460   ! The 'Published_date' data item is the special data type 'DATE' as
470   ! defined by QUERY/45.  To convert a date, such as JANUARY 1, 1975,
480   ! into its own terms, QUERY/45 uses conversion and verification
490   ! algorithms.  The algorithms let special QUERY/45 data types be used
500   ! with IMAGE/45 programs.  Refer to Appendix C for more information.
510   ! ********************************************************************
520   !
530   DEF FNEncode_date(INTEGER M,D,Y)
540         SHORT Dcode
550   !     IF NOT FNVerify_date(M,D,Y) THEN RETURN -999999
560   ! ***** The above line is normally used, but is not needed here
570         Real=INT(365.25*Y+.75)+INT(30.55*M-29.95)-2*(M>2)+D+((M>2) AND (Y/
4=INT(Y/4)))-((M>2) AND (Y/100=INT(Y/100)))
580         Dcode=Real+((M>2) AND ((Y/400=INT(Y/400)) OR (Y/1000=INT(Y/1000)))
)-((M>2) AND (Y/4000=INT(Y/4000)))-999999
590         RETURN Dcode
600         FNEND
```

Adding Data

Data is added, one entry at a time, from a program into a data base using the DBPUT statement. The data base buffer string should be PACKed first with the values you want to enter into data items.

You can add entries to manual master and detail data sets. To enter data into a detail set, each related manual master set must have an entry in which the key item value matches the corresponding key value in the detail set. When a detail set entry is added with a key value matching that of an existing detail entry, the new entry is automatically chained to the previous one. Key item values being added to manual masters must be unique.

You do not add entries to automatic master data sets. Entries are added automatically when the entry made to an associated detail set has a key value which is not found in any entry of the corresponding automatic master.

Sequence for Adding Entries

To avoid errors, before making entries into detail data sets, add all entries into related manual master data sets. This order is necessary so that the manual master entries contain the related key item values.

Programming Hint

If you are in the process of converting your application from traditional data files (PRINT#) to a data base, use READ# to get the desired values from the data files. These values can be put into the variables used in the PACKFMT statement. Then a normal PACK and DBPUT can be performed.

The DBPUT Statement

The DBPUT statement is used to add entries to a manual master or detail data set. Before DBPUT, use PACK to put the desired values in the buffer string.

DBPUT (data base name, set, 1, status array, @, buffer string)

Parameters

data base name	The same string variable containing the data base name used in the DBOPEN.
set	A string expression containing the set name, or a numeric expression containing the set number of the data set to which you want to add data.
1	Any numeric expression equal to 1. It is included for compatibility with other IMAGE systems.
status array	An array identifier for the status array (the array name followed by (*)).
@	A string expression with the value "@", "@Δ[1]" or "@;"
buffer string	A string variable serving as the data base buffer string. PACK can be used to put data for the items into the buffer string before the DBPUT is performed.

Limitations

In order to perform a DBPUT, the data base must have been opened with the DBOPEN specifying read/write access, mode 3 or mode 11. The password specified in the DBOPEN must have write access to the set to which data is being added. (This is indicated in the schema.)

In order to add an entry to a detail set, the following conditions must all be true. Otherwise, a non-zero value is returned in the first element of the status array to indicate an error.

- The data set cannot be full.
- Each related manual master set must have an entry in which its key item value matches that of the detail set.
- Each related automatic master set must either have an entry in which its key value matches that of the detail or space to allow an entry to be made automatically.
- If the data base is a multiple-volume one, the detail set and all related masters must be on-line.

In order to add an entry to a manual master set, the following conditions must be true. Otherwise, a non-zero value is returned in the first element of the status array to indicate an error.

- The set cannot be full.
- No existing entry can have the same key item value.
- If it is a multiple-volume data base, the set must be on-line.

[1] Δ signifies a blank space.

Status Array

The first element of the status array, the Condition Word, is non-zero if there is an error during the DBPUT. You should always check it after the operation. The status array contains the following information after a successful DBPUT –

Element	Value
1	0 (Condition Word–error if non-zero)
2	Number of bytes transferred from the data base buffer string
3	0
4	Record number that was accessed.
5	0
6	Unchanged from the last DBGET or DBFIND on that set
7	0
8	Unchanged from the last DBGET or DBFIND on that set
9	0
10	Unchanged from the last DBGET or DBFIND on that set

The following values for the Condition Word are possible after an unsuccessful DBPUT.

Value	Meaning
–14	DBOPEN must use mode 3 or 11
–23	You lack write access to the data set
16	Data set is full
43	Duplicate key item value (manual master sets only)
80	Data set's volume not found, or data set not created
1xx	No chain head for path xx (detail sets only). Add key item value to master set.
3xx	Automatic master for path xx is full (detail sets only)
4xx	Master set for path xx is not on-line (detail sets only)

Examples

This example adds an entry into the BOOK detail data set and related automatic masters.

```
10    !  This program illustrates DBPUT.  A new book is added into the
20    !  'BOOK' data set.  Related Automatic Master data sets are
30    !  updated if necessary. ****************************************
40    !
50    DIM Base$[9],Password$[8],Buf$[200]
60    DIM Title$[60],Author$[50],Subject$[40],Publisher$[30]
70    REAL Call_number
80    SHORT Published_date,Price
90    INTEGER Status(1:10)
100   Base$="  LIBR:F8"
110   Password$="LIBRMGR"
120   DBOPEN (Base$,Password$,3,Status(*))
130   IF Status(1)<>0 THEN Dberr       ! Branch to error routine
140 Book_fmt: PACKFMT Title$,Call_number,Author$,Subject$,Published_date,Publish
er$,Price
150   Title$="PASCAL: USER MANUAL AND REPORT"
160   Call_number=7516462
170   Author$="WIRTH,NIKLAUS"
180   Subject$="COMPUTER PROGRAMMING"
190   Published_date=FNEncode_date(1,1,1975)   ! SEE SUBPROGRAM BELOW *******
200   Publisher$="SPRINGER-VERLAG"
210   Price=6.95
```

```
220   !
230   !
240   ! ********* PACK THE BUFFER STRING AND ENTER DATA INTO THE SET:
250   !
260   PACK USING Book_fmt;Buf$
270   DBPUT (Base$,"BOOK",1,Status(*),"@",Buf$)
280   IF Status(1)<>0 THEN Dberr
290   !
300   DBCLOSE (Base$,%,1,Status(*))
310   IF Status(1)<>0 THEN Dberr
320   !
330   STOP
340 Dberr:                          ! Error routine if Status(1)<>0
350   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
360   DBCLOSE (Base$,%,1,Status(*))
370   END
380   !
390   ! ************************************************************************
400   ! The 'Published_date' data item is the special data type 'DATE' as
410   ! defined by QUERY/45.  To convert a date, such as JANUARY 1, 1975,
420   ! into its own terms, QUERY/45 uses conversion and verification
430   ! algorithms.  The algorithms let special QUERY/45 data types be used
440   ! with IMAGE/45 programs.  Refer to Appendix C for more information.
450   ! ************************************************************************
460   !
470   DEF FNEncode_date(INTEGER M,D,Y)
480           SHORT Dcode
490   !       IF NOT FNVerify_date(M,D,Y) THEN RETURN -999999
500   ! ***** The above line is normally used, but is not needed here
510           Real=INT(365.25*Y+.75)+INT(30.55*M-29.95)-2*(M>2)+D+((M>2) AND (Y/
4=INT(Y/4)))-((M>2) AND (Y/100=INT(Y/100)))
520           Dcode=Real+((M>2) AND ((Y/400=INT(Y/400)) OR (Y/1000=INT(Y/1000)))
)-((M>2) AND (Y/4000=INT(Y/4000)))-999999
530           RETURN Dcode
540           FNEND
```

This example adds entries to the BORROWER manual master data set.

```
10    !  This program adds another employee to the BORROWER data set
20    !
30    DIM Base$[9],Password$[10],Buf$[200]
40    DIM Borrower_name$[50],Employee_phone$[14]
50    INTEGER Status(1:10),Employee_no,Location
60    Base$="  LIBR:F8"
70    Password$="LIBRMGR"
80    DBOPEN (Base$,Password$,3,Status(*))
90    IF Status(1)<>0 THEN Dberr       ! Branch to error routine
100 Borrower_fmt: PACKFMT Employee_no,Borrower_name$,Location,Employee_phone$
110   !
120   ! ASSIGN VALUES TO THE 'BORROWER' ITEMS:
130   !
140   Employee_no=2411
150   Borrower_name$="JONES,GUS"
160   Location=1                          ! LOCATION IS A 'CODE' DATA TYPE AS
                                            DEFINED BY QUERY/45.  '1' STANDS
170                                       ! FOR 'MANAGEMENT'.
180   Employee_phone$="(519) 555-1234"
190   !
200   ! ENTER DATA INTO THE SET:
210   !
220   PACK USING Borrower_fmt;Buf$
230   DBPUT (Base$,"BORROWER",1,Status(*),"@",Buf$)
```

```
240   IF Status(1)<>0 THEN Dberr
250   !
260   DBCLOSE (Base$,X,1,Status(*))
270   IF Status(1)<>0 THEN Dberr
280   !
290   STOP
300 Dberr:                          ! Error routine if STATUS(1)<>0
310   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
320   DBCLOSE (Base$,X,1,Status(*))
330   END
```

Retrieving Data – DBGET

The DBGET statement is used to read an entry from the specified data set into the data base buffer string. After a DBGET operation, UNPACK can be used to put the values into program variables.

DBGET (data base name, set, mode, status array, @, buffer string, argument)

Parameters

data base name	The same string variable containing the data base name used in the DBOPEN.
set	A string expression containing the set name, or a numeric expression containing the set number of the data set from which you want to retrieve data.
mode	A numeric expression equal to 2, 4, 5 or 7 which specifies the access method (see below).
status array	An array identifier for the status array (the array name followed by (*)).
@	A string expression with the value "@", "@Δ" or "@;". It is included for compatibility with other IMAGE systems.
buffer string	A string variable serving as the data base buffer string. UNPACK may be used after the DBGET to put the values into variables.
argument	in mode 2 (serial): ignored in mode 4 (directed): a numeric expression whose value (rounded to an integer) is the record number you want to retrieve. in mode 5 (chained): ignored in mode 7 (calculated): a numeric expression or string variable containing the value of the key item you want to access. It must be the same type (numeric or string) as the key. If the argument is a string variable, its current length must be greater than or equal to the length of the key item as specified in the schema. A numeric value is converted to the proper precision, if necessary.

DBGET Modes

The mode parameter specifies the type of access method. The four modes were introduced in Chapter 2. The following table summarizes the modes and how the argument parameter is interpreted for each one.

Mode	Argument	Action	When to Use
2 – Serial	ignored	Retrieves the next non-empty record after the current one, skipping empty ones.	When retrieving all entries in a set. To start at the beginning of a set, perform a directed read to record 0. Used also with stand-alone data sets.
4 – Directed	record number	Retrieves the entry in the specified record; if it is empty, an error occurs.	For retrieving entries in reverse order along a data chain. Specify record 0 to "rewind" the data set; a subsequent serial read retrieves the first entry in the set. Is also used for random accesss when the desired record is known.
5 – Chained	ignored	Retrieves the next entry in a chain of detail entries. Use DBFIND first to find the head of (first entry in) the chain	For detail entries with the same value for a particular key item. Can also be used with a synonym chain of manual master entries.
7 – Calculated	key item value	Use with master sets to retrieve the entry with the specified key item value.	For quick retrieval of a selected entry in a master set.

Status Array

The first element of the status array, the Condition Word, is non-zero if there is an error during the DBGET. You should always check it after the operation. The status array contains the following information after a successful DBGET.

Element	Value
1	0 (Condition Word–error if non-zero).
2	Number of bytes transferred to the data base buffer string
3	0
4	Record number that was accessed
5	0
6[1]	0 for a detail set or a secondary entry in a synonym chain of master entries. For a primary entry in a master set it is the number of entries in the synonym chain.
7	0
8[1]	Record number of previous entry in the chain.
9	0
10[1]	Record number of next entry in the chain

[1] If chained mode is used on a manual master set, elements 6, 8 and 10 have different meanings. This information is useful if you are concerned with synonyms. For a **primary entry**, these elements represent the number of entries in the chain, record number of last entry and record number of first entry. For a **secondary entry**, these elements are 0, record number of previous entry and record number of next entry. Chapter 9 discusses synonyms.

The following values for the Condition Word are possible after an unsuccessful DBGET.

Value	Meaning
−21	You lack read access to the set
−31	Calculated access DBGET (mode 7) not allowed on detail set. If chained access DBGET (mode 5), detail data set has no paths
11	End-of-file in serial DBGET (mode 2); no entries following the current record
12	Negative record number in directed DBGET (mode 4) not allowed
13	Directed DBGET (mode 4) value exceeds set capacity
15	End of chain in chained DBGET (mode 5)
17	Empty record in directed DBGET (mode 4); or no matching key item value in calculated DBGET (mode 7)
18	No chain head for path xx (detail sets only).
80	Add key item value to master set.

Possible Errors

The following errors are possible during DBGET.

Value	Meaning
11	Numeric value required

Example

```
10    !  This program uses DBGET to retrieve the names of all librarians.
20    !  SERIAL mode is used.
30    !
40    DIM Base$[9],Password$[10],Buf$[200],Plant_name$[10],Librarian$[50]
50    INTEGER Status(1:10)
60    Base$="  LIBR:F8"
70    Password$="LIBRMGR"
80    DBOPEN (Base$,Password$,3,Status(*))
90    IF Status(1)<>0 THEN Dberr       ! Branch to error routine
100   !
110   ! THE FORMAT FOR 'LIBRARY' SKIPS TWO ITEMS THAT ARE NOT NEEDED:
120 Library_fmt: PACKFMT Plant_name$,120X,Librarian$,14X
130   !
140    PRINTER IS 0
150   FOR I=1 TO 19        ! CAPACITY OF LIBRARY SET
160      DBGET (Base$,"LIBRARY",2,Status(*),"@",Buf$,X)
170      IF Status(1)=11 THEN End_of_file    ! CHECK FOR LAST ENTRY
180      IF Status(1)<>0 THEN Dberr
190      UNPACK USING Library_fmt;Buf$
200      PRINT "PLANT: ";Plant_name$,TAB(25);"LIBRARIAN: ";Librarian$
210   NEXT I
220 End_of_file:           ! This line reached when last entry accessed
230   !
240   DBCLOSE (Base$,X,1,Status(*))
250   IF Status(1)<>0 THEN Dberr
260   !
270   STOP
280 Dberr:                          ! Error routine if STATUS(1)<>0
290   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
300   DBCLOSE (Base$,X,1,Status(*))
310   END
```

```
PLANT: DCD              LIBRARIAN: NELSON,ANITA

PLANT: BOISE            LIBRARIAN: BARLOW,SANDY

PLANT: GSD              LIBRARIAN: LARSEN,STACY

PLANT: DMD              LIBRARIAN: ROSS,BONNIE

PLANT: CORVALLIS        LIBRARIAN: ASHBY,BECKY

PLANT: DSD              LIBRARIAN: CHAPMAN,GAIL

PLANT: DTD              LIBRARIAN: FAGER,PAUL

PLANT: SAN DIEGO        LIBRARIAN: BABCOCK,JOHN
```

The DBFIND Statement

The DBFIND statement is used before a chained DBGET. It is used with a detail set to locate the first entry in the chain that has the key item value identified by the argument parameter. It does not change the current record pointer.

DBFIND (data base name, set, 1, status array, item, argument)

Parameters

data base name	The same string variable containing the data base name used in the DBOPEN.
set	A string expression containing the set name, or a numeric expression containing the set number of the detail set you want to access.
1	Any numeric expression equal to 1. It is included for compatibility with other IMAGE systems.
status array	An array identifier for the status array (the array name followed by (*)).
item	A string expression containing the name, or a numeric expression containing the number of the key item of the data set you want to access. The number of an item is determined by its position in the ITEM part of the Schema.
argument	A numeric expression or string variable containing a value for the key item specified by the item parameter. It must be of the same type as the key item. Numeric precision is converted, if necessary, to that of the key item. If the argument is a string variable, its length must be greater than or equal to the key item length as defined in the schema, so you may need to pad the value with blanks.

NOTE

DBFIND does not retrieve any data from the data base. It merely locates the desired record, establishes the chain count and pointers.

Status Array

The first element of the status array, the Condition Word, is non-zero if there is an error during the DBFIND. You should always check it after the operation. The status array contains the following information after a successful DBFIND.

Element	Value
1	0 (Condition word-error if non-zero.)
2	0
3	0
4	0
5	0
6	Number of entries in the chain.
7	0
8	Record number of the last entry in the chain.
9	0
10	Record number of the first entry in the chain.

The following values for the Condition Word are possible after an unsuccessful DBFIND.

Value	Meaning
−11	Improper data base specified or data base not open. Check the data base name parameter; it should not have been changed since the DBOPEN
−21	Data set not existent; or you do not have access to this set
−22	Detail set required
−31	Invalid mode parameter
−52	Item specified is not a valid key item in the master set
17	There is no chain for the specified key item value
53	Argument parameter is incompatible with key item type; or current length of a string argument is less than the string length of the key item value
5xx	Volume is on-line, but created data set xx is missing

Example

```
10    !  This program uses DBFIND and DBGET to find and retrieve all
         TITLES of BOOKS about the subject MATHEMATICS
20    !  CHAINED DBGET is used after the DBFIND
30    !
40    DIM Base$[9],Password$[10],Buf$[200]
50    DIM Title$[60],Subject$[40]
60    INTEGER Status(1:10)
70    Base$="  LIBR:F8"
80    Password$="LIBRMGR"
90    DBOPEN (Base$,Password$,3,Status(*))
100   IF Status(1)<>0 THEN Dberr      ! Branch to error routine
110   !
120   ! THE FORMAT FOR 'BOOKS' SKIPS ALL BUT TWO OF THE ITEMS:
130 Book_fmt: PACKFMT Title$,58X,Subject$,38X
140   !
150   ! LOCATE THE CHAIN HEAD:
160   !
161   Subject$[1,40]="MATHEMATICS"    ! Pad the value with blanks
```

```
170   DBFIND (Base$,"BOOK",1,Status(*),"SUBJECT",Subject$)
180   IF Status(1)<>0 THEN Dberr
190   Total=Status(6)                ! Number of entries in chain
200   !
210   PRINT "THE FOLLOWING BOOKS DEAL WITH MATHEMATICS:",LIN(1)
220   FOR I=1 TO Total               ! Total IS NUMBER OF CHAINED ACCESSES
230      DBGET (Base$,"BOOK",5,Status(*),"@",Buf$,X)
240      IF Status(1)<>0 THEN Dberr
250      UNPACK USING Book_fmt;Buf$
260      PRINT Title$
270   NEXT I
280   DBCLOSE (Base$,X,1,Status(*)) ! *** ALWAYS CLOSE DATA BASE ***
290   IF Status(1)<>0 THEN Dberr
300   !
310   STOP
320 Dberr:                          ! Error routine if STATUS(1)<>0
330   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
340   DBCLOSE (Base$,X,1,Status(*))
350   END

THE FOLLOWING BOOKS DEAL WITH MATHEMATICS:

MATHEMATICS OF PHYSICS AND MODERN ENGINEERING
PROBLEM SOLVING WITH THE COMPUTER
HANDBOOK OF MATHEMATICAL FUNCTIONS
```

Modifying Data – DBUPDATE

The current record for a detail or manual master data set (the last entry involved in a DBPUT or DBGET operation in that set) can be modified using the DBUPDATE statement. The current record is normally established by performing a DBGET to obtain the record you want to modify. Any items in the set can be modified except key items.

DBUPDATE (data base name, set, 1, status array, @, buffer string)

Parameters

data base name	The same string variable containing the data base name used in the DBOPEN.
set	A string expression containing the set name, or a numeric expression containing the set number of the data set you want to modify.
1	Any numeric expression equal to 1. It is included for compatibility with other IMAGE systems.
status array	An array identifier for the status array (the array name followed by (*)).
@	A string expression with the value "@", "@Δ" or "@;". It is included for compatibility with other IMAGE systems.
buffer string	A string variable serving as the data base buffer string. PACK can be used to put values for all the items into the string before the DBUPDATE is performed.

Limitations

In order to perform a DBUPDATE, the data base must have been opened with the DBOPEN specifying read/write access, mode 3 or mode 11. The password specified in the DBOPEN must have write access to the set being updated. (Passwords with write access are indicated in the schema.)

When updating a detail or manual master data set, the value of the key item in the buffer string must match the existing key item value in the current record; otherwise, condition word = 41 occurs.

Status Array

The first element of the status array, the Condition Word, is non-zero if there is an error during the DBUPDATE. You should always check it after the operation. The status array contains the following information after a successful DBUPDATE –

Element	Value
1	0 (Condition Word-error if non-zero)
2	Number of bytes transferred from the data base buffer string.
3	0
4	Unchanged from previous DBPUT or DBGET on that set.
5	0
6	Unchanged from previous DBPUT or DBGET on that set.
7	0
8	Unchanged from previous DBPUT or DBGET on that set.
9	0
10	Unchanged from previous DBPUT or DBGET on that set.

The following values for the Condition Word are possible after an unsuccessful DBUPDATE.

Value	Meaning
−11	Bad data base name. Don't change the first two characters
−14	DBUPDATE not allowed in access mode 8
−21	Set not found or inaccessible
−23	You lack write access to set
−24	DBUPDATE not allowed on an automatic master set
−31	Invalid mode value
−52	Bad @ parameter, must be "@;" or "@ " or "@"
17	The selected entry is empty
41	DBUPDATE will not alter a key item
80	Volume missing or set not created

Example

```
10    !  This program uses DBUPDATE to change information about
         an employee when he moves to a different location
20    !
30    DIM Base$[9],Password$[10],Buf$[200]
40    Buf$=RPT$(" ",200)                ! Set current length to 200
50    DIM Borrower_name$[50],Employee_phone$[14]
60    INTEGER Status(1:10),Employee_no,Location
70    Base$="  LIBR:F8"
80    Password$="LIBRMGR"
90    DBOPEN (Base$,Password$,3,Status(*))
100   IF Status(1)<>0 THEN Dberr       ! Branch to error routine
110 Borrower_fmt: PACKFMT Employee_no,Borrower_name$,Location,Employee_phone$
120   !
130   INPUT "Employee number of employee who has moved?",Employee_no
140   !
150   ! USE CALCULATED ACCESS TO RETRIEVE THE ENTRY FOR DESIRED EMPLOYEE:
160   !
170   DBGET (Base$,"BORROWER",7,Status(*),"@",Buf$,Employee_no)
180   UNPACK USING Borrower_fmt;Buf$
190   !
200   PRINT "CURRENT INFORMATION:",LIN(1),Borrower_name$,LIN(1),Employee_no
210   PRINT Location,LIN(1),Employee_phone$
220   INPUT "New department code?",Location
230   INPUT "New phone number?",Employee_phone$
240   !
250   ! UPDATE THE INFORMATION:
260   !
270   PACK USING Borrower_fmt;Buf$
280   DBUPDATE (Base$,"BORROWER",1,Status(*),"@",Buf$)
290   IF Status(1)<>0 THEN Dberr
300   !
310   DBCLOSE (Base$,X,1,Status(*))
320   IF Status(1)<>0 THEN Dberr
330   !
340   STOP
350 Dberr:                          ! Error routine if STATUS(1)<>0
360   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
370   DBCLOSE (Base$,X,1,Status(*))
380   END
```

Deleting Data Entries – DBDELETE

An entry can be deleted from a detail or manual master data set using the DBDELETE statement which deletes the **current record**. This is normally established by performing a DBGET to obtain the entry you want to delete. The DBOPEN must specify read/write access, mode 3 or mode 11, and its password must have write access to the set from which an entry is being deleted.

```
DBDELETE (data base name, set, 1, status array)
```

Parameters

data base name	The same string variable containing the data base name used in the DBOPEN.
set	A string expression containing the set name, or a numeric expression containing the number of the data set from which you want to delete.
1	A numeric expression equal to 1. It is included for compatibility with other IMAGE systems.
status array	An array identifier for the status array (the array name followed by (*)).

Considerations

In order to delete an entry in a manual master set, there must be no related detail entry having the same key item value. Otherwise, a non-zero value is returned in element 1 of the status array.

When deleting an entry in a detail set, the necessary changes to chains are made. When the entry is the only entry in a detail chain linked to an automatic master entry, and all other chains linked to that master entry are empty, the automatic master entry is also deleted.

When deleting entries, you may want to print each entry to serve as a journal of changes made. These can be referred to in the event that an error was made in deleting.

Current Record Pointer

After deleting an entry in a detail data set, the current record pointer is unchanged. After deletion of an entry in a master set, the location of the current record pointer is not predictable; it may or may not be the location of the current record pointer prior to the DBDELETE. The current record pointer changes when the deleted entry was a primary entry in a master synonym chain. This condition can be detected by checking element 6 of the status array when the DBGET which located the entry was performed. If element 6 is non-zero, the entry is a primary entry in a master synonym chain. If this is the case, and the value is greater than 1, the desired entry is removed from the data base and the next entry in the chain is moved into that record. The current record pointer is set to the record of the entry that moved. (This process is known as migration.)

Status Array

The first element of the status array, the Condition Word, is non-zero if there is an error during the DBDELETE. You should always check it after the operation. The status array contains the following information following a successful DBDELETE.

Element	Value
1	0 (Condition word – error if non-zero.)
2	Unchanged from previous DBGET or DBPUT on that set
3	0
4	Record number of deleted record
5	0
6	0 (If the entry deleted was a primary entry in a master synonym chain, 1 is returned instead.)
7	0
8	Unchanged from previous DBGET or DBPUT on that set[1]
9	0
10	Unchanged from previous DBGET or DBPUT on that set[1]

The following values are possible for the Condition Word after an unsuccessful DBDELETE.

Value	Meaning
−14	Data base must be opened in mode 3 or 11
−23	You lack write access to the data set
−24	Cannot delete automatic master entries
44	Cannot delete manual master entry with non-empty detail chain
80	Data set's volume not on-line; or data set not created
4xx	Master set for path xx is not on-line

1 If element 6 is non-zero, elements 8 and 10 are 0.

Examples

This example deletes selected entries from the BOOK detail data set, based on publication date.

```
10    !  This program uses DBGET and DBDELETE to find and delete all
         BOOKS published before 1965
20    !  SERIAL DBGET is used to look at all BOOK entries
30    !
40    !  **** NOTE:  After books are deleted, they should also
50    !              be deleted from the 'INVENTORY' set.  *****
60    !
70    DIM Base$[9],Password$[10],Buf$[200],Title$[60]
80    INTEGER Status(1:10),Month,Day,Year
90    SHORT Pub_date
100   Base$="  LIBR:F8"
110   Password$="LIBRMGR"
120   DBOPEN (Base$,Password$,3,Status(*))
130   IF Status(1)<>0 THEN Dberr       ! Branch to error routine
140   !
150   ! THE FORMAT FOR 'BOOKS' SKIPS ALL BUT THREE OF THE ITEMS:
160 Book_fmt: PACKFMT Title$,Call_number,90X,Pub_date,34X
170   !
180   FOR I=1 TO 89                    ! Capacity of set
190      DBGET (Base$,"BOOK",2,Status(*),"@",Buf$,X)
200      IF Status(1)=11 THEN End_of_file
210      IF Status(1)<>0 THEN Dberr
220      UNPACK USING Book_fmt;Buf$
230      CALL Decode_date(Pub_date,Month,Day,Year)  ! SEE SUBPROGRAM BELOW **
240      IF Year>=1965 THEN Nexti
250      !
260      ! THE FOLLOWING LINES EXECUTED WHEN PUBLICATION DATE IS BEFORE 1965:
270      PRINT LIN(1);"THE FOLLOWING BOOK WAS PUBLISHED BEFORE 1965:"
280      PRINT Title$
290      INPUT "DO YOU WANT TO DELETE IT? (Y or N)",Answer$
300      IF Answer$="N" THEN Nexti
310      !
320      DBDELETE (Base$,"BOOK",1,Status(*))
330      IF Status(1)<>0 THEN Dberr
340      !
350      PRINT Title$;" deleted from the data base"
360      PRINT "Its call number is ";Call_number  ! For deleting from INVENTORY
370 Nexti:  NEXT I
380 End_of_file:                  ! No more entries in the set
390   DBCLOSE (Base$,X,1,Status(*))
400   IF Status(1)<>0 THEN Dberr
410   STOP
420 Dberr:                        ! Error routine if STATUS(1)<>0
430   PRINT "UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
440   DBCLOSE (Base$,X,1,Status(*))
450   END
460   ! ************************************************************************
470   ! The 'Pub_date" data item is the special data type 'DATE' as defined
480   ! by QUERY/45. To convert a date into terms it understands, QUERY/45
490   ! uses conversion and verification algorithms, which let special
500   ! QUERY/45 data types by used with IMAGE/45 programs. Refer to
510   ! Appendix C for more information.
520   ! ************************************************************************
530   !
540   SUB Decode_date(SHORT Dcode,INTEGER M,D,Y)
550        IF (Dcode>-999999) AND (Dcode<=999999) THEN Valid
560        M=D=Y=0
570        SUBEXIT
580 Valid:  D_code=Dcode+999999
```

```
590       Y=INT((D_code-1)/365.25)
600       D_code=D_code-INT(365.25*Y+.75)
610       M=INT((D_code+31)/30)
620       Mflag=(M>2) AND (Y/4=INT(Y/4))-(Y/100=INT(Y/100))+((Y/400=INT(Y/400))
OR (Y/1000=INT(Y/1000)))-(Y/4000=INT(Y/4000))
630       IF INT(30.55*M-29.95)-2*(M>2)+Mflag>=D_code THEN M=M-1
640       D=D_code-INT(30.55*M-29.95)+(2-Mflag)*(M>2)
650       SUBEXIT
660       SUBEND
```

This example deletes all entries from a manual master set. It is included to demonstrate how to compensate for migration of entries due to synonyms. (Normally, however, you would not use DBDELETE to delete all entries, you would use the DBERASE statement.)

```
10    ! ********************************************************************
20    ! THIS SAMPLE PROGRAM ILLUSTRATES HOW TO DELETE ALL ENTRIES
30    ! IN A MANUAL MASTER DATA SET, WHICH COULD BE A PROBLEM DUE TO
40    ! 'MIGRATION' OF SYNONYMS (AS EXPLAINED IN CHAPTER 2 OF THE
50    ! IMAGE/45 PROGRAMMING MANUAL). THE PROGRAM IS NOT MEANT TO BE
60    ! RUN USING THE SAMPLE DATA BASE.
70    !
80    ! THE PROGRAM IS NOT A COMPLETE ONE; IT MERELY SHOWS THE PROGRAM
90    ! SEGMENT NECESSARY TO PERFORM THE DELETION.
100   ! ********************************************************************
110   !
120   DBGET (Base$,"SET1",4,Status(*),"@",Buf$,0)    ! 'Rewind' the data set
130   !
140   DBGET (Base$,"SET1",2,Status(*),"@",Buf$,0)    ! Serially read all entries
150   IF Status(1)=11 THEN Empty                     ! Check for serial EOF
160   IF Status(6)<>0 THEN GOSUB Primary
170   GOTO 140
180   !

190 Primary: Rec=Status(4)                           ! Save current record #
200       FOR I=1 TO Status(6)                     ! Delete synonym chain
210        DBGET (Base$,"SET1",4,Status(*),"@",Buf$,Rec) ! Current rec=primary
220        DBDELETE (Base$,"SET1",1,Status(*))    ! Delete primary/secondary
230       NEXT I
240       RETURN
250          !
260 Empty:   DBCLOSE (Base$,4,4,Status(*))         ! "SET" is empty
270   STOP
```

Obtaining Information about a Data Base – DBINFO

The DBINFO statement is used to obtain structural information about a data base. It is a tool which provides information about data items, data sets, data paths (relationships between sets), and volumes. It is useful for writing general-purpose data base programs, determining how much room is left in a set, etc. The information returned is restricted by the password/access level established when the data base is opened.

The information is returned via the data base buffer string. PACKFMT and UNPACK can be used to put the data into variables. The format of the buffer string is dependent on the DBINFO mode used. These formats are covered on the following pages and indicate the format for PACKFMT.

```
DBINFO (data base name, qualifier, mode, status array, buffer string)
```

Parameters

data base name	The same string variable containing the data base name used in the DBOPEN.
qualifier	a numeric or string expression that references a data item, data set or volume by name or by number. Its interpretation is dependent on the mode which is specified. Refer to the tables which follow.
mode	A numeric expression specifying the type of information to be returned.
status array	An array identifier for the status array (the array name followed by (*)).
buffer string	A string variable serving as the data base buffer string which contains the information after a DBINFO. PACKFMT and UNPACK must be used to put the data into program variables.

Modes

The following tables specify the various DBINFO modes, their purpose, qualifier and information returned. The heading for each table specifies the type of information it returns. The Buffer Contents and Comments columns provide information enabling you to determine the necessary format for PACKFMT. References to item, set or volume numbers are determined by their position in the schema.

Data Item Information

Mode	Purpose	Qualifier	Buffer Contents	Comments
101	Returns the number for a given data item	data item name or number	byte 1-2: data item number	integer
102	Describes a data item	data item name or number	byte 1-2: data item name 15-16: 17-18: data type Δ 19-20: item length (bytes) 21-22: dimension 23-24: 0 25-26: control number	left justified, padded with blanks (L, S, I, X) Δ indicates blank if compound item, length of subitem integers
104	Returns total number of items in the set and the number of each item.	set name or number	byte 1-2: n 3-4: data item number 2n: data item number	n = number of items all values are integers

Data Set Information

Mode	Purpose	Qualifier	Buffer Contents	Comments
201	Returns the number for the given data set	data set name or number	byte 1-2: data set number	integer positive = read access negative = read/write access (defines type of access that the password used in the DBOPEN has)
202	Describes the specified data set	data set name or number	byte 1-2: data set name 15-16: 17-18: set type Δ 19-20: entry length (bytes) 21-22: 0 23-24: 0 25-26: 0 27-28: 0 29-30: number of entries 31-32: 0 33-34: data set capacity	left justified, padded with blanks M, A or D (Δ indicates blank) integers
203	Returns total number of currently accessible sets in the data base, the set numbers, and whether the set has read or read/write access	(ignored)	byte 1-2: n 2-4: ±data set number 2n: ±data set number	n = number of accessible sets positive = read access negative = read/write access (defines type of access that the password used in the DBOPEN has) arranged in same order as SET part of schema. All values are integers.
204	Identifies all accessible sets which contain the specified data item	data item name or number	same as Mode 203	

Data Path Information

Mode	Purpose	Qualifier	Buffer Contents	Comments
301	Identifies all paths defined for the specified set	data set name or number	byte 1-2: n 3-4: data set number 5-6: key item number 7-8: 0 9-10: data set number 11-12: key item number 13-14: 0 . . .	n = number of paths These are repeated for each path. If qualifier is a detail set, set number is for master. If qualifier is a master, set number is for a detail. Item numbers identify items in detail set. Path designators are presented in order of their appearance in the schema All values are integers
302	Identifies a key item for specified data set	master data set name or number OR detail data set name or number	byte 1-2: key item number 3-4: 0 byte 1-2: number of first key item[1] 3-4: data set number	integer integers

Volume Information

Modes 401, 402, 403 and 404 return information related to multi-volume data bases. Multi-volume data bases are covered in Chapter 9, Advanced Data Base Concepts.

Mode	Purpose	Qualifier	Buffer Contents	Comments
401	Identifies a volume number for a given set	data set name or number	byte 1-2: ±volume number	integer positive = volume not specified by a VOLUME DEVICES ARE negative = volume is specified by a VOLUME DEVICES ARE
402	Returns a volume name	volume number	byte 1-8: volume name	Volume 0 is the root file volume left justified, padded with blanks
403	Identifies all the volumes for a given data base excluding the root file volume	(ignored)	byte 1-2: n 3-4: ±volume number . . . 2n: ±volume number	n = number of volumes if n = 0, the entire data base is located on the root file volume. positive = volume not specified by a VOLUME DEVICES ARE negative = volume is specified by a VOLUME DEVICES ARE
404	Identifies all the data sets on a given volume	volume number	byte 1-2: n 3-4: ±data set number . . . 2n: ±data set number	Volume 0 is root file volume n = number of data sets positive = read access negative = read/write access (defines type of access that the password used in the DBOPEN has)

1 If the detail set is a stand-alone detail, mode 302 returns 0 in bytes 1 and 2.

Item/Set Relationship Information

Mode	Purpose	Qualifier	Buffer Contents	Comments
501	Returns the item length and its position in the set	numeric expression equal to item#*128+set#	byte 1-2 item length 3-4 offset in set	integers Offset specifies how far from the beginning of the set the item is (as defined by its position in the schema), in bytes.

Status Array

The first element in the status array, the Condition Word, is non-zero if there is an error during the DBINFO. You should always check it after the operation. The status array contains the following information after a successful DBINFO.

Element	Value
1	0 (Condition word – error if non-zero.)
2	Number of bytes transferred to the data base buffer string
3	0
4	Unchanged
5	The mode used in the latest DBOPEN
6	402 (identification number for the DBINFO statement)
7	Line number of DBINFO statement
8	0
9	Mode number of DBINFO
10	Integer – for system use only

The following values for the Condition Word are possible after an unsuccessful DBINFO.

Value	Meaning
−11	Bad data base name. Don't change the first two characters
−21	Bad item, set or volume reference. It is either non-existent or inaccessible. Volume reference must be numeric.
−31	Improper mode value or type
50	Buffer string is too short for requested data

Example

```
10    ! ***************************************************************
20    !  This program uses DBINFO to provide information about those
30    !  sets to which the current password has access
40    ! ***************************************************************
50    !
60    DIM Base$[9],Password$[8],Buf$[66],Type$[2],Name$[16]
70    INTEGER Status(1:10),Sets(1:32),I,J,Entries,Cap
80    Base$="  LIBR:F8"
90    Password$="LIBRMGR"
100   DBOPEN (Base$,Password$,3,Status(*))
110   IF Status(1)<>0 THEN Dberr
120   !
130   ! Use DBINFO to get the set number of those to which password has access
```

```
140   !
150   DBINFO (Base$,"",203,Status(*),Buf$)
160   IF Status(1)<>0 THEN Dberr
170   !
180   ! UNPACK the information
190   !
200   UNPACK USING Numb_of_sets;Buf$        ! UNPACK only the first integer
210 Numb_of_sets:    PACKFMT I
220   REDIM Sets(1:I)
230   UNPACK USING Set;Buf$
240 Set:     PACKFMT 2X,Sets(*)
250   !
260   ! Use DBINFO  to return the space available in each data set
270   !
280   PRINT LIN(2),"SET NAME";TAB(20);"TYPE";TAB(40);"CAPACITY";TAB(60);"ENTRIES
"
290   PRINT LIN(1)
300   FOR J=1 TO I
310      DBINFO (Base$,ABS(Sets(J)),202,Status(*),Buf$)
320      IF Status(1)<>0 THEN Dberr
330      UNPACK USING Setinfo;Buf$
340 Setinfo:   PACKFMT Name$,Type$,10X,Entries,2X,Cap
350      PRINT Name$,Type$,Cap,Entries
360   NEXT J
370   STOP
380 Dberr:  PRINT " UNEXPECTED DATA BASE ERROR ";Status(1);" IN LINE ";Status(7)
390   DBCLOSE (Base$,X,1,Status(*))
400   END
```

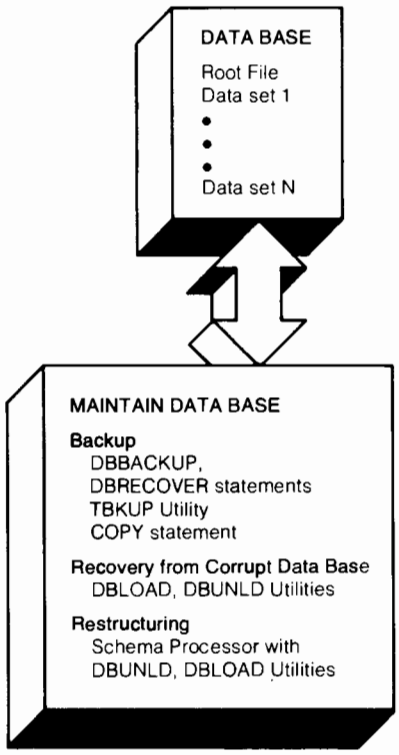
```
SET NAME          TYPE              CAPACITY          ENTRIES


AUTHOR             A                  89                67
CALL_NUMBER        A                  89                70
SUBJECT            A                  53                44
TITLE              A                  89                68
LIBRARY            M                  13                8
BORROWER           M                  79                62
BOOK               D                  89                69
INVENTORY          D                  193               148
```

Chapter 5
Backup and Recovery

Introduction

The IMAGE/45 DBM system is designed to maintain and ensure the integrity of IMAGE/45 data bases. There is the possibility, however, of losing data or structural information due to hardware failure, user error, operating system error or some external cause such as a power failure. When data or structural information is lost, the data base is said to be **corrupt**. Therefore, it is essential to adopt regular backup procedures in anticipation of possible trouble. A convenient way to accomplish this is to end each application program with an option to back up. If a failure does occur, the backup version can be used to return the data base to the state it was in when the backup was made. Since changes made between backups are not logged, you may also want to implement a method for logging changes.

IMAGE/45 provides two statements, DBBACKUP and DBRECOVER with which backup and recovery routines can be written. A BASIC utility program using these statements, TBKUP, is also provided for interactive backup and recovery.



Backup File

DBBACKUP creates one or more backup files, type BKUP, which can contain the root file and all or some of data set files. These files can either be on a different mass storage medium than the data base, or on the same one. It is best, however, to place the backup files on a separate medium that is not needed for normal operations to ensure security of the backup version. The backup file can also be on one or more tape cartridges.

Backup Methods

There are three methods for backing up your data base. They are –

- Use the TBKUP Utility program supplied with your DBMS
- Write your own programs using DBBACKUP and DBRECOVER
- Use the COPY statement on data set files and the root file.

Frequency of Backup

It is recommended that you back up your data base at regular intervals. The frequency is dependent on the number of changes made, how critical the changes are and whether or not changes made between backups are logged. DBOPEN returns the number of changes made since the last complete backup in element 8 of the status array. The largest value DBOPEN can return is 2047; if more changes are made, 2047 is still the value returned.

Considerations for Partial Backup and Recovery

It is strongly recommended that you back up all the data sets and the root file of a data base. This ensures that the entire data base can be recovered in the event of a loss. However, very large sets or those which contain non-critical or rapidly changing data may make total backup impractical or unnecessary. Here are some things to consider when making a partial backup –

- If some of the sets are backed up but the root file is not, the backup may also be corrupt when recovery is attempted after a failure. This is especially true if all master and detail data sets which are related are not backed up at the same time.
- If some of the sets and the root file are backed up, when recovery is attempted the sets that were not backed up are flagged as "unrecognized". Upon recovery, those sets must be purged using the PURGE statement and then recreated with DBCREATE. This destroys data they contained.

Using the TBKUP Utility Program For Backup

To make a backup of your data base using TBKUP, follow these steps –

1. To load the program from the Utility tape, execute LOAD "TBKUP[msus]". (The msus is necessary if TBKUP is not on the current default mass storage device.)
2. When you run the program, select the backup option. You are then asked to enter the following information –
   - the select codes of a multiple-volume data base. Press CONTINUE if it is not a multiple-volume data base.
   - data base name, followed by the msus of the root file if it is not on the current default mass storage device
   - password
   - maintenance word, if needed
   - the name of the backup file and the msus where you want it stored.
   - a protect code if you want to protect the backup file

   Press CONTINUE to enter each item of information.

NOTE

The media on which the data base resides cannot be write protected while running TBKUP.

The DBBACKUP Statement

The DBBACKUP statement creates one or more backup files to contain all or part of the data base. It can only be executed from a program, not from the keyboard.

DBBACKUP data base name [; maintenance word] [, set list] TO file specifier; return variable, return string

DBBACKUP return variable, return string

The first syntax is used to set up the desired backup operation; it is the primary syntax. The second syntax is used for all subsequent DBBACKUP statements until the backup process is complete. The return variable and return string are used to indicate a complete backup, flag a corrupt data base or prompt the program user to prepare for a subsequent DBBACKUP statement.

Parameters

data base name — A 4-character string variable containing the data base name. The msus of the root file should follow the name if the root file is not on the current default mass storage device.

maintenance word — The same string expression, (of which the first six characters are used), specified in the DBCREATE. It provides data base security.

set list — To back up selected sets, include set numbers separated by commas. An asterisk as the first item in the list (*) causes the root file to be backed up. If no set list is specified, the entire data base is backed up, including the root file.

file specifier — A string expression containing the file name and optional msus of the backup file. If no msus is specified, the backup file is created on the current default mass storage device.

return variable — A simple numeric variable used to pass information back to the program during a DBBACKUP. Possible values and their meanings are –

1. Backup completed normally.
2. Another data set, whose name is in the return string, is about to be backed up. Do a secondary DBBACKUP using the current value of the return string.
3. – **4.** Not used.
5. Another backup destination file is needed. Put its file name and msus, if desired, into the return string and do a secondary DBBACKUP, but do not insert the medium.
6. Insert (or mount) the medium indicated when the return variable was 5 and do a secondary DBBACKUP. Don't alter the value of the return string.
7. Insert (or mount) the source volume indicated in the return string, execute a VOLUME DEVICES ARE statement for this volume and do a secondary DBBACKUP.

return string	A string variable at least 16 characters long used to pass information back to the program during a DBBACKUP. It is used for creation of secondary DBBACKUPs. Data in the string returned is left justified.

Special Considerations

The number of physical records needed to back up a data base is the sum of the following items –

- the size of each data set (DSET) file in the data base
- the size of the root file
- one additional record for each data set
- one additional record for the root file
- one record for each backup file created **after** the first one, as discussed below.

If all of these records can fit in one place on one medium, the entire data base is backed up into one file. If all of these records do not fit in one place, DBBACKUP indicates that another destination backup file is needed by placing a 5, then a 6 in the return variable. Secondary entries of DBBACKUP resume the backup process.

If you are backing up a multiple-volume data base, DBBACKUP indicates that another source volume is need by placing a 7 in the return variable. The source volume should be mounted and a VOLUME DEVICES ARE statement must be executed. When the return variable is 2, check to see that the volume for the set is on-line. If it is not, mount it, execute a VOLUME DEVICES ARE statement, then execute a secondary DBBACKUP. Multiple-volume data bases are covered in Chapter 9.

NOTE

Do not remove any discs during the backup operation unless the return variable indicates that this is necessary.

Possible Errors

The following errors are possible during a DBBACKUP.

Value	Meaning
211	Improper data base name
219	Out of user read/write memory for DBBACKUP
221	Data set not created; cannot be backed up
224	Return variable has been altered; or no primary DBBACKUP statement has been executed to set up the backup
225	Improper utility version number in the root file
229	DBBACKUP not allowed while the data base is open
233	Required data set or root file not on-line

Example

```
10    ! **** This is a general-purpose backup program which can be
20    ! **** used to back up any data base. It uses DBBACKUP *****
30    !
40    DIM Base$[9],Maint$[8],Setlist$[150],Com$[16],Bkup$[16],Vda$[160],A$[1]
50    LINPUT "ENTER DATA BASE NAME  ( e.g. LIBR or BASE:F8 )",Base$
60    INPUT "IS IT A MULTI-VOLUME DATA BASE? (Y or N)",A$
70    IF A$="Y" THEN INPUT "ENTER THE MSUS OF EACH DEVICE ON WHICH THE DATA BASE
 IS FOUND (e.g. :F8:F9:F4 )",Vda$
80    INPUT "ENTER MAINTENANCE WORD (PRESS CONTINUE IF NO MAINT. WORD)",Maint$
90    LINPUT "ENTER SET LIST (OR PRESS CONTINUE FOR ENTIRE DATA BASE)",Setlist$
100   LINPUT "ENTER NAME FOR BACKUP FILE (e.g. BKUP:T OR BACK2 )",Bkup$
110   PRINT LIN(2);"** DBBACKUP **";LIN(1),"BACKUP FILE NAME:   ";Bkup$
120   !
130   DBBACKUP Base$;Maint$,Setlist$ TO Bkup$;Code,Com$ ! Set up the backup **
140   PRINT LIN(2);"Root file backed up"
150   !
160 Continue:    ON Code GOSUB Done,C2,C3,C4,C5,C6,C7   ! Check 'Code'; may loop
170   DBBACKUP Code,Com$                                ! Secondary DBBACKUP
180   GOTO Continue
190   !
200 C2:           ! Another data set about to be backed up
210   PRINT LIN(2);"Data set ";TRIM$(Com$);" about to be backed up"
220   RETURN
230 C3:           ! Not used
240 C4:           ! Not used
250 C5:           ! Another backup destination file is needed
260   PRINT LIN(2);"Data base does not fit in one backup file; need another one"
270   PRINT "Enter the new name when asked, but DO NOT mount the new volume yet"
280   LINPUT "ENTER THE NEW BACKUP NAME (e.g. BKUP2:T or TEMP)",Com$
290   RETURN
300 C6:           ! Mount the new backup volume
310   PRINT LIN(2);"Mount the new backup volume; the new backup file is: ";Com$
320   DISP "PRESS CONTINUE WHEN READY"
330   PAUSE
340   DISP ""
350   RETURN
360 C7:           ! Mount a new source volume
370   PRINT LIN(2);"Volume ";TRIM$(Com$);" cannot be located. Please mount it an
d press CONTINUE when ready."
380   PAUSE
390   VOLUME DEVICES ARE Vda$
400   RETURN
410 Done:    END
```

Backup Using the COPY Statement

The COPY statement can be used to backup the data set files and root file of a data base. The copy must be to a disc. If you have a multiple-volume data base, each set should be copied to a disc which has the same volume label as the disc on which is resides.

CAUTION

TO PERFORM A SUCCESSFUL COPY OF DATA BASE FILES, YOUR SYSTEM 45 MUST HAVE REVISION C OF A CERTAIN OPERATING SYSTEM ROM. THIS CAN BE DETERMINED BY PERFORMING A COPY OPERATION ON ANY ONE OF THE DATA BASE FILES, THEN PERFORMING A CAT OPERATION. IF THE TYPE OF THE COPIED FILE HAS CHANGED FROM ROOT OR DSET TO OPRM, THEN YOU DO NOT HAVE THE PROPER REVISION. CALL YOUR HP CUSTOMER ENGINEER AND HE WILL REVISE YOUR MACHINE.

Example

This program copies all the files for the sample data base.

```
10    ! **** This program copies the root file and all data sets *****
20    ! **** The COPY must be to a disc. Discs must have the same
30    ! **** label if it is a multiple-volume data base.
31    !
50    DATA LIBR,LIBR01,LIBR02,LIBR03,LIBR04,LIBR05,LIBR06,LIBR07,LIBR08
60    FOR I=1 TO 9
70       READ File$
80       COPY File$&":F8" TO File$&":F4","BOOKS"      ! MAINTENANCE WORD ***
90    NEXT I
100   END
```

Using the TBKUP Utility Program for Recovery

To use TBKUP to recover from a data base error, follow these steps –

1. Load the program as in Step 1 of backup.
2. When you run the program, select the recovery option. You are then asked to enter the following information –
   - the msus of each volume in a multiple-volume data base. Press CONTINUE if it is not a multiple-volume data base.
   - the name of the data base.
   - the name of the backup file followed by its msus if it is not on the current default mass storage device.
   - the mass storage unit specifier of the disc on which you want to place the root file
   - no other version of the data base can exist on the same device as the new one. You must stop the program and use DBPURGE if this is the case. If not, enter YES.
3. After the root file and data set files are recovered, enter YES if there are QUERY/45 related files that you wish to recover.

The DBRECOVER Statement

Should it become necessary, the DBRECOVER statement is used to restore the data base using the backup file. The data base is returned to the state it was in when it was backed up. Only those sets that were backed up can be recovered with DBRECOVER. Others may or may not be useable. DBRECOVER can only be executed from a program, not from the keyboard.

Before using DBRECOVER, a DBPURGE should be executed to purge everything that was backed up.

```
DBRECOVER backup file [ON msus    ]; return variable, return string
                      [ON volume  ]
DBRECOVER return variable, return string
```

The first syntax is used to set up the desired recovery operation; it is the primary syntax. The second syntax is used for all subsequent DBRECOVER statements until the recovery process is complete. The return variable and return string are used to indicate a completed recovery or to prompt the program user to prepare for a subsequent DBRECOVER statement.

Parameters

backup file
: A string expression containing the name of the backup file specified in DBBACKUP.

msus
volume label
: A string expression containing the msus or volume label of the medium where you want to put the new (recovered) root file. If it is omitted, the current default mass storage device is used.

return variable
: A simple numeric variable used to pass information back to the program after a DBRECOVER. Possible values and their meanings are –

1. Recovery completed normally.
2. Another data set, whose file name is in the return string, is ready to be recovered. Do a secondary DBRECOVER using the current value of the return string.
3. Not used.
4. Incomplete backup data. Check to make sure that backup files are used in the same order as they were created. A secondary DBRECOVER should be executed.
5. Not used.
6. A new source backup file is needed. The name of the backup file is in the return string. The msus for the backup file should be added to the name in the return string if it is not on the current default mass storage device, and a secondary DBRECOVER performed.
7. A new destination volume is needed; its volume name is in the return string. Ensure that a VOLUME DEVICES ARE has been executed and perform a secondary DBRECOVER.

return string — A string variable 16 characters in length used to pass information back to the program during a DBRECOVER. It is used in the creation of secondary DBRECOVERs. Data in the string is left justified.

Special Considerations

If only selected sets were backed up, remaining sets are unchanged by a DBRECOVER. If you backed up selected sets and the root file, remaining sets are considered "uncreated". You must DBPURGE them, then recreate them with DBCREATE. However, complete backups are strongly recommended.

Possible Errors

The following errors are possible during a DBRECOVER.

Value	Meaning
219	Out of user read/write memory for DBRECOVER
223	Improper backup file; return variable has been altered or no primary DBRECOVER statement has been performed.
224	Backup sets have been mounted in the wrong order. The DBRECOVER process must be started from the beginning
229	DBRECOVER not allowed while the data base is open

Example

```
10    ! **** This is a general-purpose recovery program which can be
20    ! **** used to recover a backed up data base. It uses DBRECOVER ****
30    !
40    DIM Com$[16],Bkup$[16],Vda$[160],A$[1],Scode$[12],Ncode$[12]
50    LINPUT "ENTER NAME OF BACKUP FILE (e.g. BKUP:T OR BACK2)",Bkup$
60    INPUT "IS THE DATA BASE BEING RECOVERED A MULTI-VOLUME ONE? (Y or N)",A$
70    IF A$="Y" THEN INPUT "ENTER MSUS OF EACH DEVICE WHICH CONTAINS A DATA BASE
 VOLUME (e.g. :F8:F9:F4)",Vda$
80    LINPUT "ENTER THE MSUS OF THE ROOT FILE DEVICE (e.g. :F8 or:D)",Scode$
90    !
100   Vda$=Vda$&Scode$              ! In case the root file volume was omitted
110   VOLUME DEVICES ARE Vda$
120   !
130   DBRECOVER Bkup$ ON Scode$;Code,Com$
140 Continue:  ON Code GOSUB Done,C2,C3,C4,C5,C6,C7
150   DBRECOVER Code,Com$
160   GOTO Continue
170   !
180 C2:          ! Another data set about to be recovered.
190   PRINT LIN(2);"Recovering data set ";TRIM$(Com$);"."
200   RETURN
210 C3:          ! Not used
220 C4:          ! Incomplete backup set.
230   PRINT LIN(2);"   The backup set is incomplete.  Please begin again and be
sure that the backup volumes are mounted in the correct order."
240   BEEP
250   STOP
260 C5:          ! Not used
270 C6:          ! New source backup file needed.
```

```
280   PRINT LIN(2);"The next source backup file: ";TRIM$(Com$);", is now needed.
  Please mount it and enter its select code."
290   LINPUT "ENTER SELECT CODE OF THE NEXT BACKUP FILE (e.g. :F or :T )",Ncode$
300   Com$=Com$&Ncode$
310   RETURN
320 C7:           ! New recovery destination volume needed.
330   PRINT "MOUNT THE NEW RECOVERY DESTINATION VOLUME ";TRIM$(Com$);" AND PRESS
 CONTINUE"
340   PAUSE
350   VOLUME DEVICES ARE Vda$
360   RETURN
370 Done:         ! All done.
380   PRINT LIN(2);"DATA BASE ";TRIM$(Base$);" HAS BEEN SUCCESSFULLY RECOVERED."
390   STOP
```

Recovery of a Corrupt Data Base That Has No Backup

If data or structural information is lost, the data base is corrupt. In the event that there is no backup version, it may be possible to salvage all or most of the data by using the DBUNLD and DBLOAD utility programs. (These programs are discussed in Chapter 7.) Follow these steps to use DBUNLD and DBLOAD to salvage data in a corrupt data base.

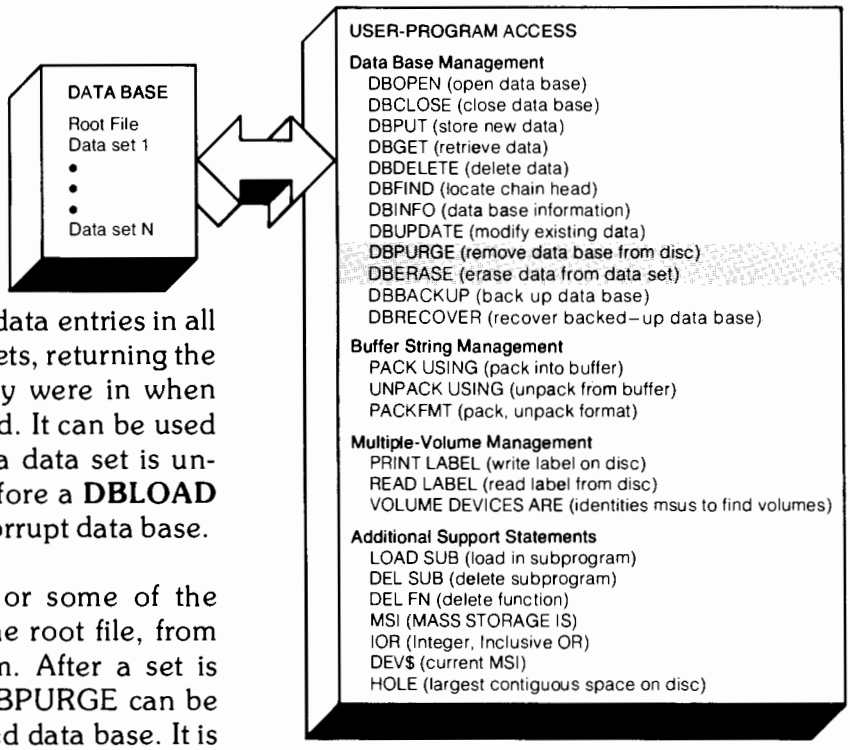
1. Run DBUNLD, reading data entries in **serial** fashion
2. Use DBERASE to erase all data sets, or use a DBPURGE if ERROR 226 occurs. If DBPURGE is necessary, the root file and data sets must be recreated with the Schema Processor and DBCREATE.
3. Run DBLOAD to load the data back into the data base.

There is another method by which data can be read from a corrupt data base. It involves opening the data base with DBOPEN mode 8, which returns +94 in the Condition Word of the status array, indicating that a corrupt data base was successfully opened. In general, most of the data entries can be read using DBGET, using either serial mode or directed mode. Retrieval can continue in this fashion until invalid information is found in the root file. At this point, remaining sets may or may not be readable.

Chapter 6

Erasing Data Base Information

There are two large-scale erasures that can be performed on a **closed** data base –



DBERASE erases all the data entries in all or some of the data base sets, returning the data sets to the state they were in when **DBCREATE** was executed. It can be used when the information in a data set is unwanted. It is also used before a **DBLOAD** when recovering from a corrupt data base.

DBPURGE removes all or some of the data set files, including the root file, from the mass storage medium. After a set is purged, its data is lost. DBPURGE can be used to purge an unwanted data base. It is also used before a **DBRECOVER** is performed.

CAUTION

THE PURGE STATEMENT SHOULD NOT BE USED TO PURGE A DATA SET; DBPURGE SHOULD ALWAYS BE USED. PURGE DOES NOT CHANGE ROOT FILE INFORMATION, SO THE ROOT FILE WILL INDICATE THAT THE SET IS STILL PRESENT.

The DBERASE Statement

The DBERASE statement erases all data entries and related path information from one or more sets in a data base.

DBERASE data base name [; maintenance word][, set list | * | , msus | , volume specifier]

Parameters

data base name	A string variable containing the data base name. The msus of the root file should follow the name as part of the string if it is not on the current default mass storage device.
maintenance word	The same string expression (of which only the first 6 characters are used) that was specified when the data base was created.

The final parameter has four possible variations. The first two are used to erase only selected sets. The second two parameters are used primarily with multiple-volume data bases and tell the system to erase the sets that reside on the specified medium. If one of the four final parameters is not included, DBERASE erases all of the sets in the data base.

set list	A string expression which contains set numbers separated by commas. It lets you erase only selected sets.
*	A string expression containing an asterisk ("*"); it tells the system to erase all sets that are available.
msus	A string expression of the form- :device type [select code[,controller address | 9885 unit code[,unit code]]] It tells the system to erase all sets on that medium.
volume specifier	A string expression containing the volume label preceded by a comma (",volume label"). If you specify a volume, it must be on-line. This is done with a VOLUME DEVICES ARE statement.

Considerations

When a DBERASE is performed on a detail data set, the associated entries in automatic master sets are **not** erased. Hence, to avoid errors, all related automatic master sets should also be erased with DBERASE. Additionally, all related manual master sets must be on-line. When a master data set is erased, all of the path information about its links to detail data sets is erased also. Therefore, to avoid errors, all related detail sets should be erased with DBERASE.

Possible Errors

The following errors are possible during DBERASE.

Value	Meaning
208	Needed volume not on-line
209	Operation not allowed on tape
212	Set number in the set list cannot be larger than the set count
218	Volume name is not part of the data base
220	Improper or illegal maintenance word
221	Data set not created
225	Improper utility version number in root file
226	Corrupt data base; must purge and redefine. Purge the root file and run the Schema Processor
227	Corrupt data base; all sets require erasure. When erasing a detail data set, ensure that all related master data sets are on-line.
229	Operation not allowed when the data base is open
230	Improper set list or duplicate set numbers in the set list

Examples

```
10    ! This program erases all of the data base
20    !
30    !  *** IF YOU RUN IT, YOU WILL ERASE THE SAMPLE DATA BASE !!! ***
40    !
50    MASS STORAGE IS ":F8"
60    DBERASE "LIBR";"BOOKS"
70    END
```

```
10    ! This program erases sets 1,2,3,4 and 7
20    ! This is the BOOK detail set and all related automatic masters
30    !
40    !  *** IF YOU RUN IT, YOU WILL ERASE THE SAMPLE DATA BASE !!! ***
50    !
60    DBERASE "LIBR:F8";"BOOKS","1,2,3,4,7"
70    END
```

The DBPURGE Statement

The DBPURGE statement purges all or some of data set files, including the root file of the data base.

DBPURGE data base name [; maintenance word] [, set list | , * | , msus | , volume specifier]

Parameters

data base name	A string variable containing the data base name. The msus of the root file should follow the name if the root file is not on the current default mass storage device.
maintenance word	The same string expression (of which only the first 6 characters are used) that was specified when the data base was created.

The final parameter has four possible variations. The first two are used to purge only selected sets. The second two parameters are used primarily with multiple-volume data bases and tell the system to purge the sets that reside on the specified medium. If one of the four final parameters is not included, DBPURGE purges all of the sets in the data base along with the root file.

set list	A string expression which contains set numbers separated by commas. This lets you purge only selected sets.
*	A string expression containing an asterisk ("*"); it tells the system to purge all sets not yet purged, then purge the root file. No error is given if a set was purged with PURGE.
msus	a string expression of the form – :device type [select code [, controller address | 9885 unit code [, unit code]]] It tells the system to purge all sets on that medium.
volume specifier	A string expression containing the volume label preceded by a comma (", volume label"). It tells the system to purge all sets on that medium. If you specify a volume, it must be on-line. This is done with a VOLUME DEVICES ARE statement.

Possible Errors

The following errors are possible during a DBPURGE.

Value	Meaning
208	Needed volume not on-line
209	Operation not allowed on tape
212	Set number in the set list cannot be larger than the set count
218	Volume name is not part of the data base
220	Improper or illegal maintenance word
221	Data set not created
225	Improper utility version number in root file
226	Corrupt data base; must purge and redefine. Purge the root file and run the Schema Processor
229	Operation not allowed when the data base is open
230	Improper set list or duplicate set numbers in the set list

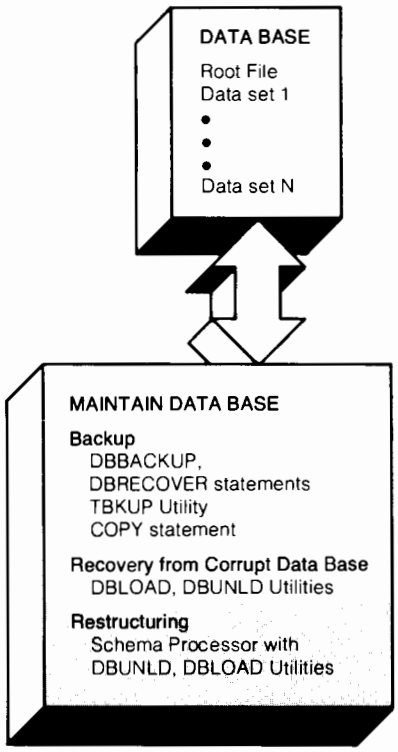
Example

```
10    ! This program purges a data base
30    !
40    !  *** IF YOU RUN IT, YOU WILL PURGE THE SAMPLE DATA BASE !!! ***
50    !
60    DBPURGE "LIBR:F8";"BOOKS"
70    END
```

Chapter 7
Restructuring a Data Base

Introduction

After you have used a data base for a period of time, you may find that it is not serving its purpose as well as it used to. Perhaps there are items or sets which are no longer needed. Perhaps there is additional information you are using which is not in the data base, but should be. At this point, it would be desirable to define a new structure for the data base but still use the valid data.



IMAGE/45 provides the capability to define a new structure for (restructure) an existing data base, then transfer the data from the old structure to the new one. This capability is provided by two utility programs, DBLOAD and DBUNLD. DBUNLD "unloads" all or part of the data into a file which contains raw data but no structural information. DBLOAD then transfers the data into the new structure, automatically reestablishing linkage information. These operations can be fairly time-consuming, but keep you from having to transfer the data yourself. DBLOAD and DBUNLD are also used for recovering from a corrupt data base which was not backed up; this concept was covered in Chapter 5.

DBUNLD does not transfer data entries from automatic master sets. Automatic masters are regenerated automatically by DBLOAD.

Reordering a Detail Data Set

DBUNLD and DBLOAD also enable you to reorder the entries in a detail data set by using a **chained unload**. The reordering uses the set's primary key item (the first key item specified in the set definition in the Schema) and places all entries that have the same key item value (a chain) together. This allows future DBGETs to be performed faster.

You can reorder a detail set in conjunction with other design changes as listed on the next page, or you can do it separately. If you only want to reorder the entries in detail data set for better performance and have no design changes to make, follow these steps.

1. Run DBUNLD to unload the desired set, selecting a **chained unload**.
2. Use DBERASE to erase the set.
3. Run DBLOAD to put the reordered entries back into the set.

Design Changes

Following are changes that can be made to the data base structure.

- Adding, changing or deleting passwords and password numbers.
- Changing the read/write access lists for data sets.
- Changing data set capacities.
- Adding new data items.
- Removing data items not used as key items.
- Rearranging the order of the items in a data set. However, in a manual master set, the key item cannot move.
- Changing the length of string items. (However, a key item cannot be made shorter.)
- Changing the name of an item or set and all references to it.
- Changing a manual master to a detail set, and vice versa.
- Adding or deleting data sets.

Changing an automatic master to a manual master data set **cannot be done**. Similarly, neither a manual master nor a detail set can be changed to an automatic master.

If the changes you wish to make to the data base structure are fairly significant, it may be simpler to read the data you want to keep from the data base and use PRINT# to store it into data files. You can then implement the new design and populate this data base using READ# to get the data out of the files.

Restructuring Procedure

To restructure a data base, these steps should be followed.

1. Run the DBUNLD program to transfer the data in one or more sets to an "unload" file.
2. Purge the old data base using DBPURGE.
3. Redesign the data base and type the new Schema text.
4. Run the Schema Processor to create the new root file.
5. Use DBCREATE to create the new data sets.
6. Run the DBLOAD program to load the data into the new data base.

NOTE

If you are adding a manual master data set to the data base, an extra step must be taken between steps 5 and 6. After the data base is created, the new manual master must be filled with data before DBLOAD is used to reload the rest of the data.

The DBUNLD Program

The DBUNLD (data base unload) program creates an unload file which contains all or part of the data in the data base. This data is used for restructuring a data base or recovering a corrupt data base (corrupt data bases are covered in Chapter 5).

Loading the Program

To load and run the DBUNLD program, execute –

```
LOAD "DBUNLD[msus]",1
```

The msus must be included if the program is not on the current default mass storage device.

Information to Enter

The following information must be entered at the beginning of the DBUNLD program –

- data base name and msus of the root file if it is not on the current default mass storage device
- maintenance word
- password with read/write access to all sets

Program Options

DBUNLD uses softkeys to enable you to select options. Options are displayed at the bottom of the CRT; the desired option is selected by pressing the SFK (Special Function Key) in the corresponding location.

The DBUNLD program gives you the option to unload either the entire data base (all sets except automatic masters, which are regenerated automatically by DBLOAD) or a selected set. You are asked to enter the set name in the latter case. If you are unloading each set separately, DBUNLD must be run multiple times. If **a detail or manual master set isn't unloaded, data in it is lost**.

NOTE

If you are restructuring a data base and want to reorder items or sets, change the number of items or sets, change item types or change capacities, unload the data base one set at a time.

The HARD COPY OPTION softkey alternates between ON and OFF. If selected, this option provides information about the number of entries in the set and the number unloaded.

You are also asked to specify either a chained or a serial unload. Serial mode uses the physical order of the entries and is faster; it must be used for recovering from a corrupt data base. Chained mode is recommended for detail sets. It can improve access times in the new data base by placing chained detail data entries closer together.

The CHECKREAD softkey alternates between ON and OFF. If set to on, checkread verification is performed.

Finally, you are asked to enter a unique name for the unload file being created to hold the unloaded data. The msus should be included if the file doesn't belong on the current default mass storage device.

If you are unloading a multiple-volume data base, DBUNLD requests volumes automatically as they are needed.

The "Unload" File

DBUNLD creates a data file, called an "unload" file, to contain the unloaded data. If a data base is unloaded one set at a time, each set has its own unload file. If there is enough contiguous space on the storage medium, the unloaded data is put into one file. If there is not enough contiguous space, DBUNLD creates as many of the unload files as it needs. If this is the case, you are asked to supply names for the additional unload files.

Read Errors

If you are unloading a data base using the serial option and a read error 87 or 88 occurs, the system attempts to recover as many entries as possible. If an entry is lost, the unload process stops after the unloading of that set. You can attempt to unload remaining sets one set at a time.

The DBLOAD Program

The DBLOAD program puts the unloaded data back into a new or unloaded data base. If the data base was unloaded in serial mode, all entries stay in the same order. If it was unloaded in chained mode, related detail entries are stored adjacent to each other, thus improving efficiency.

Loading the Program

To load and run the DBLOAD program execute –

```
LOAD "DBLOAD[msus]",1
```

The msus must be included if the program is not on the current default mass storage device.

Information to Enter

The following information must be entered at the beginning of the DBLOAD program –

- data base name to be loaded and the msus of the root file if it is not on the current default mass storage device.
- maintenance word
- password

Program Options

First, the program gives you a HARD COPY ON or OFF option, which provides an audit trail if selected. You also have the option to erase the data base. This is a necessary step for recovering, so indicate that the program should do it if it hasn't been done. If you are restructuring a data base, erasure is not necessary, since the sets are newly created.

You then have the option to select CHECKREAD ON, then load either the entire data base or a single set. If you specify a single set, you are asked to give its new name, and corresponding set number in the original (unloaded) data base.

NOTE

If you want to reorder items or sets, change the number of items of sets, change item types or change capacities, load the data base one set at a time.

Next, you are asked to enter the name of the unload file. Its msus should be specified if it is not on the current default mass storage device.

New Item Order

If you specified loading of a single data set, you can choose to reorder the items in the set. Items are reordered in terms of their position in the structure of the original data base. Enter the new order for the item numbers, separating them with commas. For newly defined items, enter 0. New items get a value of 0 or the null string.

For example, suppose you had a detail set with five items and you want to reverse their order and add an item at the end. These changes are reflected in the new schema. To specify the new order, enter 5,4,3,2,1,0

Considerations

If you are loading into a data base with a different number of sets than the original, you must unload and load one set at a time.

Data items being loaded must be of the same type (numeric or string) as the corresponding item in the new data base. If the precision of a numeric item is different and digits are lost, a message is printed to that effect. Strings are truncated or padded with blanks as necessary. The dimension value of compound items need not be equal. If the new value is less, extra items are deleted. If the new value is greater, new subitems get a value of 0 (for numeric items) or the null string.

Example

The NOP library has found that their data base is not serving them as well as it had. They determine that these three changes need to be made –

- Add an automatic master set 'PUBLISHED_DATE' which has a path to the 'BOOK' detail set. This is necessary because many searches are being done based on a book's age
- Change the capacity of 'INVENTORY' detail set to 307
- Add a new item to the 'BOOK' detail set called 'PAGE_COUNT'

Their first step is to rewrite and RE-SAVE the Schema text file. The following lines are added to the schema text listed in Chapter 3.

```
185  !                          PAGE_COUNT,    I;

442  !     NAME:                PUBLISHED_DATE,A(/5);
443  !     ENTRY:               PUBLISHED_DATE(1);
444  !     CAPACITY:            89;

655  !                          PAGE_COUNT,
```

The following lines are changed; the change is underlined –

```
660  !                            PUBLISHED_DATE (PUBLISHED_DATE),

770  !     CAPACITY:              307;
```

The next step is to run the DBUNLD program four times, unloading the manual master and detail sets one set at a time. After this is completed, the data base is purged with DBPURGE.

The next step is to run the Schema Processor, using the revised schema text to create a new root file. After this is done, DBCREATE is used to create the new data sets.

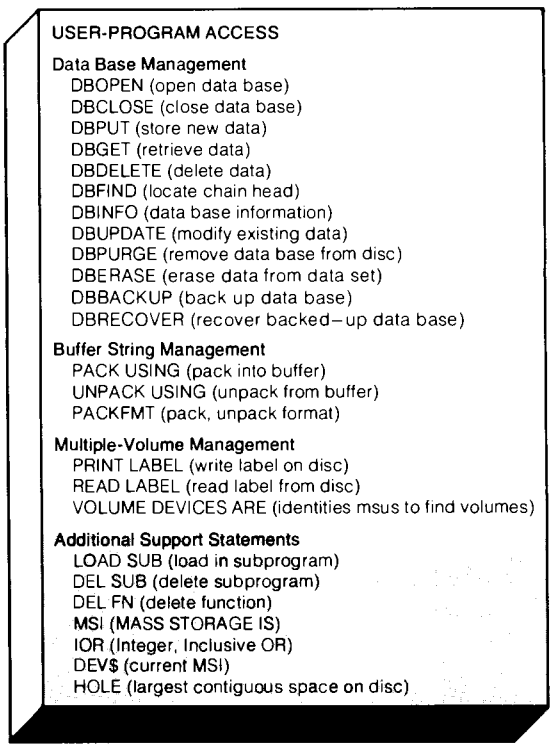
The final step is to run the DBLOAD program four times to load the data back into the data base. This should be done one set at a time. When the 'BOOK' set is reloaded, a new item order should be specified so that the new item can be added. The new order to specify is –

1,2,3,4,0,5,6,7,

Chapter 8
Miscellaneous Statements

The IMAGE/45 ROM provides seven miscellaneous statements and functions. They are –

- MSI statement – abbreviation for MASS STORAGE IS
- DEV$ function – returns the current default mass storage device
- IOR function – inclusive OR, bit-by-bit
- HOLE function – returns the size of the largest amount of contiguous available space on the current default mass storage device
- LOAD SUB statement – loads one or more subprograms, appending them to the program in memory.
- DEL SUB statement – deletes one or more subprograms; the first one deleted must be a subroutine subprogram
- DEL FN statement – deletes one or more subprograms; the first one deleted must be a function subprogram



The MSI Statement

The MSI statement is an abbreviated MASS STORAGE IS statement and selects the default mass storage device. When it is stored into a program, it is expanded to MASS STORAGE IS.

```
MSI msus
```

The DEV$ Function

The DEV$ (device) function returns a string of up to eight characters whose value is the current default mass storage device.

```
DEV$
```

The information returned is structured as follows –

For tape cartridge – :device type select code (:T15 for example)

For flexible disk – :device type select code, 9885 unit code (:F8,0 for example)

For hard disc – : device type select code, controller address, unit code (:C12,0,1 for example)

The following letters are used to indicate the various devices –

Letter	Device
T	Tape Cartridge
F	9885 Flexible Disk
Y	7905A Removable Platter
Z	7905A Fixed Platter
C	7906A Removable Platter
D	7906A Fixed Platter
P	7920A Disc Pack
X	7925A Disc Pack

Example

```
10   ! This program shows various results of DEV$
20   MASS STORAGE IS ":T"
30   PRINT DEV$
40   MASS STORAGE IS ":F"
50   PRINT DEV$
60   MASS STORAGE IS ":C"
70   PRINT DEV$
80   END

:T15
:F8,0
:C12,0,0
```

The IOR Function

The IOR function performs a bit-by-bit inclusive-or operation on two integer expressions.

IOR (numeric expression, numeric expression)

The numeric expressions must be in integer range, from −32 768 through 32 767.

Example

```
10    ! ******************************************************
20    ! This program shows sample results of the IOR function
30    ! ******************************************************
40    !
50    DATA 45,112,4,12,5,5,0,0,0,6
60    PRINT " A"," B","IOR(A,B)",LIN(1)
70    FOR I=1 TO 5
80       READ A,B
90       PRINT A,B,IOR(A,B)
100   NEXT I
110   END
```

```
 A                  B                 IOR(A,B)

 45                 112                125
 4                  12                 12
 5                  5                  5
 0                  0                  0
 0                  6                  6
```

The HOLE Function

The HOLE function returns a value equal to the largest number of contiguous, free physical records (256-bytes) on the default mass storage device.

HOLE

Restrictions

The HOLE function can't be included in some I/O statements, such as OUTPUT.

The largest possible value for HOLE is 32 767. If the medium has a larger free area than this, HOLE still returns 32 767.

Example

```
10    ! **** This program uses HOLE to determine if there is enough
20    ! **** room for a 77-record file. If so, it is created.
30    !
40    MASS STORAGE IS ":F8"
50    IF HOLE<77 THEN Too_small
60    CREATE "SCRTCH",77
70    STOP
80 Too_small:    PRINT "NOT ENOUGH ROOM FOR FILE, USE ANOTHER MEDIUM"
90    END
```

The LOAD SUB Statement

The LOAD SUB statement loads one or more subprograms into memory from a program file (PROG), adding them to the end of the program currently in memory. Both subroutine subprograms (SUB) and function subprograms (DEF FN) can be loaded. If necessary, the added lines are renumbered.

LOAD SUB file specifier [, starting line number [, increment]]
[; starting subprogram [, ending subprogram]]

Parameters

file specifier	The file specifier of the file which contains the desired subprograms. It must have been created by STORE and cannot contain any binary routines or references to binary routines.
starting line number	Causes the subprograms to be renumbered, starting with that number. If it is omitted or is 0, they are renumbered starting with the last line number + 10. It must be greater than the last line of the program in memory; otherwise, ERROR 41 occurs.
increment	Defines the line number increment for renumbering the subprograms. If omitted, it defaults to 10.
starting subprogram	The number of the subprogram with which you want to start loading. The subprograms in the file are numbered implicitly, with 1 being the first subprogram after the main program. If it is omitted, 1 is default.
ending subprogram	The number (position in the file) of the last subprogram you want to load. All subprograms between and including the starting and ending subprograms are loaded. If omitted, it defaults to the last subprogram. The ending subprogram number must be greater than or equal to the starting subprogram number.

Considerations

If any of the lines that are loaded have a line label that is the same as a line already in memory, no error occurs. If, however, a reference to one of the doubly defined labels is made, an error occurs at that time.

LOAD SUB cannot be used to replace a subprogram in memory by loading lines with the same numbers. The subprogram to be replaced must be deleted first.

If you are using an ON ERROR statement, it does not trap some of the errors that can be caused by LOAD SUB. The only LOAD SUB errors that ON ERROR can trap are 19, 38, 39 and 41.

LOAD SUB Errors

The following errors are possible during a LOAD SUB.

Value	Meaning
2	Not enough user read/write memory to load the subprograms
19	Invalid parameter value. Only starting line number can be 0; none of the parameters can be negative; starting subprogram number must be less than ending one
38	I/O function not allowed.
39	FN reference not allowed.
40	Renumbering error: starting line number within existing lines or renumbering would cause line number > 32 766
41	Starting line number > ending line number. The specified line number must be greater than the last line of the current program
56	File name is undefined
58	Improper file type; must be PROG
59	Unexpected end-of-file
65	Incorrect data type; bad information in the file
8x	Miscellaneous mass storage errors
207	Binaries not allowed in LOADSUB file

Example

```
10   ! ***************************************************************
20   ! These program lines illustrate various versions of the
30   ! LOAD SUB statement.  In order to run the lines, there must
40   ! be a file called 'ROTATE' on ':F9' which contains at least
50   ! four subprograms.
60   ! ***************************************************************
70   !
80   LOAD SUB "ROTATE:F9"          ! Load all subprograms in the file,
                                     appending them to the current program
90   LOAD SUB "ROTATE:F9",700,5    ! Renumber subprograms to start
                                     with 700, by 5's
100  LOAD SUB "ROTATE:F9";2,4      ! Load the second, third and fourth
                                     subprograms in the file.
110  END
```

The DEL SUB Statement

The DEL SUB statement is used to delete one or more subprograms from memory. The specified subprogram must be a subroutine subprogram (beginning with SUB).

DEL SUB subprogram name [TO END]

TO END causes all subprograms following the one specified to be deleted.

The DEL FN Statement

The DEL FN statement is used to delete one or more subprograms from memory. The specified subprogram must be a function subprogram (beginning with DEF FN).

DEL FN subprogram name [TO END]

TO END causes all subprograms following the one specified to be deleted.

NOTE

The DEL FN should not be entered into a program when SPACE DEPENDENT mode is set; it does not syntax correctly. Additionally, when a GET operation is performed on a file that contains a DEL FN statement, an error will occur if SPACE DEPENDENT mode is set.

DEL SUB/DEL FN Errors

Error 7 is the error that can occur during a DEL SUB or DEL FN operation.

Example

```
10    ! ***** This is the main program  ******
20    !        Still the main program
30    !
40    DEL SUB Three
50    DEL FNFive TO END
60    !
70    END      ! End of the main program
80    !
90    SUB One
100   SUBEND
110   SUB Two
120   SUBEND
130   SUB Three
140   SUBEND
150   SUB Four
160   SUBEND
170   DEF FNFive
180   FNEND
190   SUB Six
200   SUBEND
```

New listing after the program above is run –

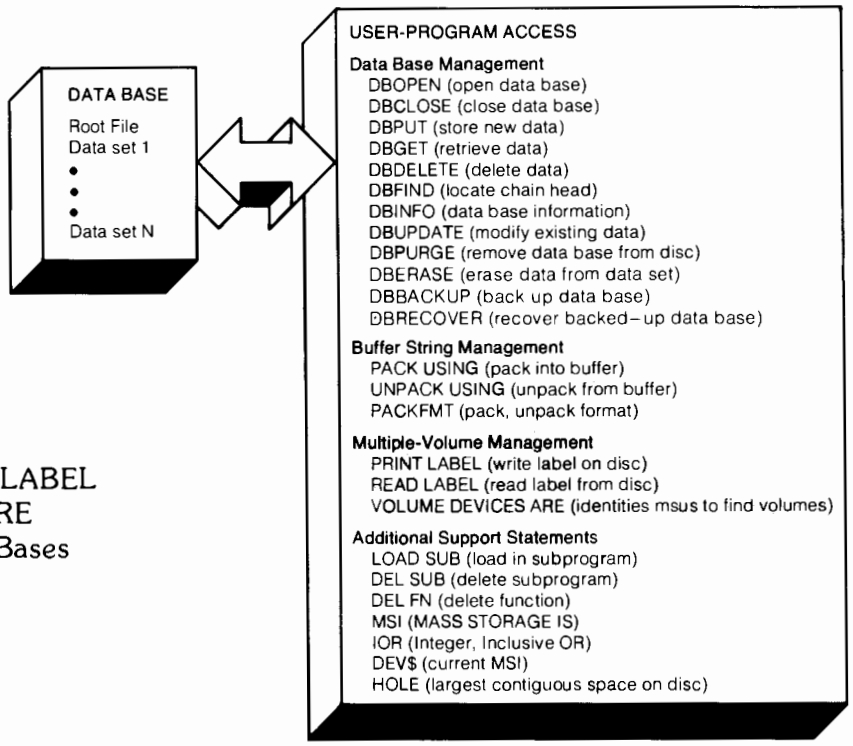
```
10    ! ***** This is the main program  ******
20    !        Still the main program
30    !
40    DEL SUB Three
50    DEL FNFive TO END
60    !
70    END      ! End of the main program
80    !
90    SUB One
100   SUBEND
110   SUB Two
120   SUBEND
150   SUB Four
160   SUBEND
```

Chapter 9
Advanced Data Base Concepts

Introduction

The data base operations discussed in the preceding chapters provide all the capability needed to use a data base. There are other operations and concepts which enable you to perform more advanced data manipulation. The concepts covered in this chapter are –

Volumes
PRINT LABEL, READ LABEL
VOLUME DEVICES ARE
Multiple-volume Data Bases
Disc Initialization
Synonyms
Memory Management
Record Structures
Root File Organization



Volumes

IMAGE/45 enables you to give any disc a name, known as a **volume label**. Labels can also be read. PRINT LABEL and READ LABEL enable these capabilities. Since discs can be accessed by their label, it is not necessary to keep track of the msus of each disc being accessed. With the VOLUME DEVICES ARE statement, you tell the system the msus of each disc you want to access. It reads and tabulates the label from each one, putting the information into a **volume device table**, then recognizes these as the current **on-line volumes**. Up to four volumes can be on-line at a time.

The PRINT LABEL Statement

The PRINT LABEL statement writes a label onto the disc in either the current mass storage device or the device specified after ON. Additionally, the volume device table is updated if the device is an entry in it.

PRINT LABEL volume label [ON msus]

Parameters

volume label	A string expression of up to eight characters. A null string erases the current label. Any colon (:), comma (,), space, NUL (CHR$(0)) or DEL (CHR$ (127)) is removed before the label is printed.
msus	A string expression of the form – : device type [select code [, controller address | 9885 unit code [, unit code]]]

NOTE

To prevent certain errors, never rename a disc that is part of a data base. The root file keeps information about which volume labels are part of the data base.

The READ LABEL Statement

The READ LABEL statement has two forms. The first form reads the label off of the current default mass storage device or off the specified device. Additionally, if the specified device is in the volume device table, the table is updated to ensure it contains the label. The second form fills a string array with the contents of the volume device table.

READ LABEL string variable [ON msus]
READ LABEL string array identifier

Parameters

string variable	After a READ LABEL, this simple string contains the label of the current mass storage device, or of the device specified after ON.
msus	A string expression of the form – : device type [select code [, controller address | 9885 unit code [, unit code]]]
string array identifier	After a READ LABEL, the array contains the contents of the volume device table. This array must be a one-dimensional string array. Ten is the greatest number of elements needed (maximum number of entries in the volume device table), though fewer can be used. The maximum length for each element is 17 characters (longest possible entry).

Possible Errors

The following errors are possible during a READ LABEL.

Value	Meaning
13	Array not dimensioned
16	Array must be one-dimensional
18	Array elements must have at least 16 characters

Example

```
10    ! This program illustrates PRINT LABEL and READ LABEL
20    !
30    MASS STORAGE IS ":F8"
40    VOLUME DEVICES ARE ":F8:F4"
50    PRINT LABEL "CURRENT" ON ":F4"     ! Labels medium in ":F4"
60    !
70    ! ************ To determine label of current mass storage device, use:
80    !
90    READ LABEL Disk_label$
100   PRINT "LABEL ON CURRENT MASS STORAGE DEVICE IS ";Disk_label$
110   !
120   ! ************ To determine which volumes are currently on-line, use:
130   !
140   DIM Vol_table$(10)                 ! String array
150   READ LABEL Vol_table$(*)
160   PRINT LIN(2);"THE FOLLOWING DEVICES AND VOLUMES ARE CURRENTLY ON-LINE:"
170   PRINT Vol_table$(*)
180   !
190   END


THE FOLLOWING DEVICES AND VOLUMES ARE CURRENTLY ON-LINE:
DB2:F8,0
CURRENT:F4,0
```

The VOLUME DEVICES ARE Statement

The VOLUME DEVICES ARE statement tabulates on-line volumes. It clears any existing table and sets up the volume device table to contain each specified msus and its related label. It must be used whenever a new data base volume is mounted during execution of a program which uses the data base.

VOLUME DEVICES ARE msus list

Parameters

msus list	A string expression containing up to ten mass storage unit specifiers.

Possible Errors

The following errors can occur during a VOLUME DEVICES ARE.

Value	Meaning
52	Improper mass storage unit specifier in list
73	Improper device type specified in one of the mass storage unit specifiers
209	Operation not allowed on tape

There are five error conditions that are not recognized by VOLUME DEVICES ARE. If any of these occurs on a device, that device **is not** included in the volume device table, but no error message is given. An error may occur, however, when you try to access a set on a drive that caused the error. The possible errors are –

71 – disc interface power off
72 – incorrect controller address, or controller power off
74 – drive missing, or drive power off
80 – door open or disc out
82 – device not present

If any other error occurs, the volume device table is cleared and an error message given.

Example

```
10    ! ********************************************************
20    ! The following program lines illustrate various versions
30    ! of the VOLUME DEVICES ARE statement. This statement
40    ! should be used at the beginning of a program involving
50    ! a multiple-volume data base to indicate which volumes
60    ! are on-line. It should also be executed any time a
70    ! volume is moved.
80    ! ********************************************************
90    !
100   MASS STORAGE IS ":F8"
110   VOLUME DEVICES ARE DEV$       ! Current mass storage device
120   !
130   !
140   DIM Vda$[160]                 ! Contains all the msus's
150   INPUT "ENTER THE MASS STORAGE UNIT SPECIFIER OF EACH DATA BASE VOLUME (e.g
. :F8:F9:C5,4)",Vda$
160   VOLUME DEVICES ARE Vda$
170   END
```

Multiple – Volume Data Bases

The ability to give any disc a label, then read that label enables a data base to reside on more than one disc. This enables data bases to be very large, since they need not be contained on one disc. It also enables sensitive data to be stored on a separate volume which can be removed from general access.

There are a few limitations to multiple-volume data bases. Each data set must be contained on a single volume; in other words, a data set cannot span volumes. Another limitation is that the maximum number of labeled volumes a data base can span is 23. Additionally, the root file may be on another disc, which is **never** accessed by using a volume label, but instead by its msus. Data sets may also reside on this disc. A third limitation is that a maximum of four volumes can be on-line at any one time.

The three statements which enable volume access are READ LABEL, PRINT LABEL and VOLUME DEVICES ARE. They were discussed earlier in this chapter.

Specifying Volumes

The volume on which a set is to reside is declared in the schema. If you want a set to be on the same disc as the root file, no volume name may be specified for it in the schema.

The Root File

The root file is an essential part of data base operations and must be accessible at all times while the data base is open. When a data base is opened, the root file must either be in the current default mass storage device, or you must tell the system where to find it by appending its mass storage unit specifier to the data base name in the DBOPEN statement.

CAUTION

THE ROOT FILE MUST REMAIN IN THE SAME MASS STORAGE DEVICE WHILE THE DATA BASE IS OPEN. NEVER MOVE ITS MEDIUM UNLESS THE DATA BASE HAS BEEN CLOSED.

Accessing Multiple Volumes

A VOLUME DEVICES ARE statement must be executed at the beginning of the program and should include all devices that contain volumes that you want to use. It must also be executed each time a volume is moved. Any attempt to access a set in a volume that is not on-line results in an error.

When accessing a set in a multiple-volume data base, make sure that all related sets (those linked in a master/detail relationship) are on-line. This is necessary in order to perform an update.

There are three operations which may have to be handled differently in a multiple-volume environment: DBCREATE, DBERASE and DBPURGE. Since it may not be possible to have all the volumes of a data base on-line at the same time, these operations may require several repetitions of the statement in conjunction with exchanging of discs and execution of a VOLUME DEVICES ARE statement.

CAUTION

IN A MULTIPLE-VOLUME DATA BASE, NONE OF THE VOLUMES USED SHOULD BE WRITE PROTECTED UNLESS NO WRITE OPERATIONS ARE GOING TO BE PERFORMED. THIS HELPS PREVENT UPDATE ERRORS.

Exchanging Volumes

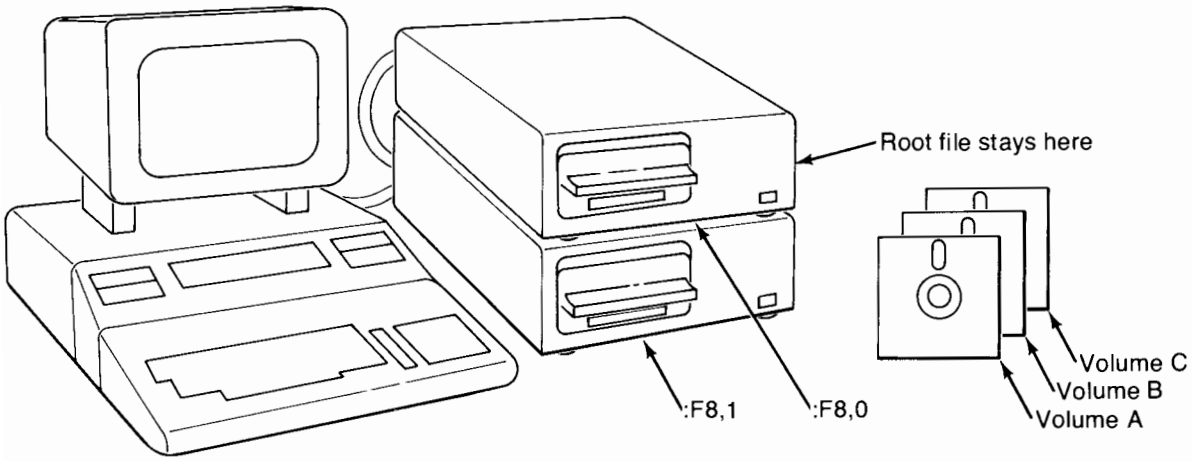
There are two important things to remember when moving discs in a multiple-volume situation –

- Before removing a volume from a drive, ensure that all data has been written to it. This is accomplished either by executing a DBCLOSE mode 4, or opening the data base with DBOPEN mode 3.
- After exchanging volumes, excute another VOLUME DEVICES ARE statement to inform the system of which volumes are on-line.

Remember, the disc containing the root file must not be moved while the data base is open.

Example

Suppose you have a System 45 with two HP 9885 disk drives. You have a data base having three volumes, A, B and C with the root file on a fourth disk.



When the data base is opened, :F8,0 should be the current default mass storage device, or should be appended to the data base name parameter in the DBOPEN. At that time, and any time a different volume is placed into :F8,1, the following statement should be executed –

```
VOLUME DEVICES ARE ":F8,0:F8,1"
```

Here is an example of using DBCREATE for a multiple-volume data base. Assume that it is being run on the system described above.

```
10    ! ************************************************************************
20    ! This program illustrates DBCREATE for a multiple-volume data base.
30    ! This data base is called 'SAMP'.
40    !
50    ! According to the schema, the root file and data sets reside on
60    ! four different disks, as detailed below -
70    !    - ROOT FILE VOLUME (NO LABEL) : root file, set "1"
80    !    - VOLUME 'A'                  : sets "2" and "3"
90    !    - VOLUME 'B'                  : set "4"
100   !    - VOLUME 'C'                  : sets "5" and "6"
110   !
120   ! Assume that the three disks are already labeled.
130   !
140   ! ************************************************************************
150   !
160   DIM Vda$[10]
170   Vda$=":F8,1:F8,0"
180   PRINT "INSERT ROOT FILE VOLUME IN ':F8,0' AND VOLUME 'A' IN ':F8,1'"
190   PRINT "PRESS CONTINUE WHEN READY"
200   PAUSE
210   VOLUME DEVICES ARE Vda$
220   !
230   DBCREATE "SAMP","1,2,3"      ! Create sets 1,2 and 3
240   !
250   PRINT "INSERT VOLUME 'B' IN ':F8,1'; PRESS CONTINUE WHEN READY"
260   PAUSE
270   VOLUME DEVICES ARE Vda$
280   !
290   !
300   DBCREATE "SAMP",",B"          ! Create sets on volume 'B'
310   !
320   PRINT "INSERT VOLUME 'C' IN ':F8,1';  PRESS CONTINUE WHEN READY"
330   PAUSE
```

```
340   VOLUME DEVICES ARE Vda#
350   !
360   DBCREATE "SAMP","*"          ! Create those sets not yet created
370   !
380   END
```

Disc Initialization

To maximize performance of data base operations, 3 is the recommended interleave factor for HP 9885 flexible disks. In general, this number may cause slower initialization and non-data base data transfers (PRINT# and READ#). For estimated initialization times, refer to the Mass Storage ROM manual.

Synonyms

When an entry is placed into a master data set, its record number is determined through a process called hashing. Hashing uses the value of the key item in an algorithm which produces the record number. When a key item value produces a record number that already has an entry in it, it is known as a **synonym** of the existing entry. The system places the synonym in the next available record and creates a pointer from the original entry to the synonym. The first entry that hashes to a record is called a primary entry; synonyms are called secondary entries. Related synonyms are linked together in a synonym chain.

When the primary location of an entry is filled with a synonym of another entry, the synonym is moved to an empty record, the pointers are updated and the new entry is placed in its primary location. This process is known as **synonym migration**.

Synonyms can be undesirable because they cause greater access time. You can determine the frequency of synonyms in a master set by checking element 6 of the status array after performing a DBGET on each entry in the set. Element 6 contains the number of synonyms for that key value. If most of the operations return a small value or 0 in element 6, then synonyms are not a problem.

There are three steps that can be taken to reduce synonyms –

- Make sure the capacity is a prime number
- Make sure the master set is not filled to more than 75% capacity. If necessary, increase the capacity by restructuring the data base.
- Examine the key item values, especially of strings. In general, repeated combinations of letters may produce the same record number. For example, all the following strings hash to the same record –

  XY
  XYAAAA
  XYAAAAAAAA
  XYABAB
  XY343444446767

  It may be desirable to change the key item, or alter the values in some way before they are stored in the data base.

Memory Management

If you are opening and closing a number of data bases in the same program, or if you are backing up a number of data bases, use them in order from largest to smallest. The reason for doing this is because DBOPEN uses memory which is not deallocated by DBCLOSE. The amount of memory used depends on the size of the data base. After a data base is closed, **this** memory can be used by later DBOPENs, so a smaller data base can fit in the memory previously used by a larger one.

Conserving Read/Write Memory

DBOPEN uses part of the value area of read/write memory when executed in the main program. There is, however, a special area in memory which can be used when the DBOPEN is executed in a subprogram and which is not part of user read/write memory. The maximum size of this special area is 12 000 bytes, but some of it may be used by the system for other purposes.

If it is necessary to conserve read/write memory, this special area of memory can be utilized by executing any DBOPEN in a subprogram. Remember that the system may use some of this memory for other purposes. This condition, along with excessively large DBOPEN blocks, can cause ERROR 219, meaning the DBOPEN must be in the main program. The following section explains the memory usage of DBOPEN.

DBOPEN Memory Usage

The first DBOPEN of a data base allocates three separate areas of memory –

1. A common area for use by all data bases
2. Global block and data set control block
3. Local block and data set dynamic table and private data base buffer

Any subsequent DBOPEN of the same data base (prior to a DBCLOSE, mode 1) allocates only a block of type 3, using previous blocks 1 and 2. A DBOPEN of a different data base allocates blocks of type 2 and 3.

The sizes of the three blocks are determined as follows –

Global block – 70 bytes
+20 bytes per item
+20 bytes per set
+(20 bytes +4*(# of items+ # of key items) per set

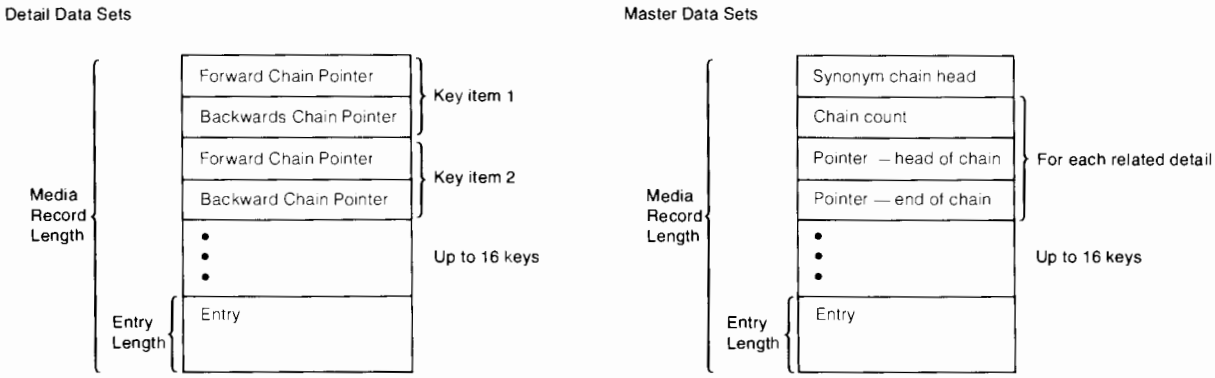
Local block – 58 bytes
+12 bytes per set
+private buffer size

The private buffer size can be determined using the following table –

Largest Media Record in Data Base (in bytes)	Private Buffer Size (in bytes)
6 – 256	512
258 – 512	768
514 – 768	1024
770 – 1024	1280

Record Structures

The following diagrams illustrate how a media record is structured.



The synonym chain head (first three words) of a master data set record can have one of the following configurations –

Empty Record[1]	Primary Entry	Secondary Entry
–1	Synonym Chain Count (≥1)	0
Pointer to Next Empty Record	Pointer – 1st member of chain	Pointer to Next Member of Chain
Pointer to Previous Empty Record	Pointer – last member of chain	Pointer to Previous Member of Chain

1 This also applies to empty detail data set records.

Root File Organization

The root file is a file created by the Schema Processor and is of type ROOT. It contains all of the essential information concerning a particular data base. It contains information about the passwords, items, sets, and paths that are defined in the data base. There are seven major sections in the root file. They are as follows –

1. **Password Table**
   Contains the passwords defined in the Schema along with the maintenance word, version numbers, and the global block length.
2. **Set Read/Write Specification Table**
   This table contains the read and write access specifications, by set.
3. **Volume Label Table**
   Contains, by set, the volumes defined in the data base, the Set Creation table (bit map of the sets created) and the Set Erasure table.
4. **Data Base Global Information**
   This is 64 bytes of specific information (such as data base name, address, pointers, item/set count, and length).
5. **Item Table**
   This table contains 20 bytes for each item which describe the item name, length, type and points to the next item in the item synonym chain.
6. **Set Table**
   This table contains 20 bytes per set and defines the set number, name, length, type, data set control block pointer, and points to the next set in the set synonym chain.
7. **Data Set Control Blocks**
   There is one Data Set Control Block (DSCB) for each set which defines the paths, items, and information pointers for that set.

The following diagram illustrates root file organization.

Password Table	256 bytes
Set Read/Write Specifications	256 bytes
Volume Label Table/Set Creation Table	256 bytes
Data Base Global Information	64 bytes
Item Table	20 bytes/item
Set Table	20 bytes/set
Data Set Control Blocks	24 to 596 bytes/set

The first three physical records of the root file are called the Header; the information in this section shouldn't change after the data base has been created.

The last four sections contain information that can be changed by the data base statements; the size of these sections is relative to the number of items, sets and paths. The Data Base Global information contains offsets into the Item and Set tables, and the Set table has a pointer to its corresponding Data Set Control Block.

Appendix A
Glossary

Terms

The following are terms which relate to data base concepts and programming.

Access Method – the method by which data entries are retrieved. There are four types used by IMAGE/45: serial, directed, chained and calculated.

Attribute – the identifying characteristics of an entity. An attribute is represented by a **data item**.

Automatic Master – a data set which contains one item, the key item. It must be related to at least one detail set. When a new value for a key is added to a related detail set, a new entry is automatically added to the master.

Backup File – the file created by the DBBACKUP statement.

Buffer String – see Data Base Buffer String.

Calculated Access – a method for retrieving an entry in a master data set. It involves specifying a value for the key item.

Capacity – the maximum number of entries a set can hold. It is specified in the schema and can be an integer from 1 to 32 767. A master set capacity should be a prime number.

Chain – a series of pointers which links together all detail data set entries having the same value for a particular key item. The system automatically maintains all chains.

Chained Access – a method of retrieving detail entries in which the next entry in a specific chain is retrieved.

Current Record – the last entry in each data set read or written by a DBGET or DBPUT. DBOPEN sets all current record pointers to 0.

Data Base – a collection of logically related files that contain data and structural information, and are stored on a disc.

Data Base Buffer String – a string variable used to pass data between a data base and the program. PACK, UNPACK and PACKFMT are used to put data into, retrieve data from and format the buffer string.

Data Base Name – a string of 1 to 4 characters. The first character must be a letter; the rest of the characters can be letters or digits.

Data Chain – see chain.

Data Entry – a collection of values for the items in a data set which describe a particular occurrence of the entity. All data is transferred to and from the data base on an entry basis. Also known as a record.

Data Item – the smallest accessible data element which represents an attribute of an entity. Data items are given values and can correspond to a program variable. Also known as a field.

Data Item Name – a string of 1 to 15 characters beginning with a letter. The characters can be uppercase and lowercase letters, digits and the underscore character (_). Nationalized characters can be included if you have a local-language keyboard.

Data Path – see Path.

Data Set – a collection of data items which describes an entity or relationship between entities. All data entries in a data set are together stored in a separate file. There are two types of data sets, master and detail.

Data Set File – a file containing all the entries for a particular data set.

Data Set Name – a string of 1 to 15 characters. The characters can be uppercase and lowercase letters, digits and underscore character (_). Nationalized characters can be included if you have a local-language keyboard.

Detail Data Set – a set containing data items which describe an entity.

Dimension – an integer used in the schema to specify the number of subitems in a compound item.

Directed Access – a method of retrieving a data entry in which the record number is specified.

Entity – a thing such as a person or place. A data base keeps information about entities.

Field – see data item.

File – a collection of related information stored on a mass storage medium. There are three types of files associated with IMAGE/45 data bases: data set files, root files and backup files.

Hashing – a process which manipulates a key item value using some algorithm, producing a record number.

Item – see Data Item

Item Name – see Data Item Name.

Key Item – an item in a detail set which links it to a master set by a path. Also known as a search item.

Maintenance Word – a string expression (of which only the first 6 characters are used) specified in the first DBCREATE for a data base which is used as a protect code. It is needed for DBCREATE, DBERASE, DBPURGE and DBBACKUP.

Manual Master Set – a master data set which can contain items in addition to the key item. A manual master can stand alone; it need not be linked to a detail. Entries must be explicitly added or deleted.

Master Data Set – a data set which contains a key item and is generally used as an index into one or more detail data sets.

Media Record – an entry and its related chain pointers.

msus – mass storage unit specifier. Any string expression of the form –
: device type [select code [, controller address | 9885 unit code [, unit code]]]

Password – a 1 to 8 string of ASCII characters > 32 and < 128, excluding semicolons, which is used to gain access to a data base.

Password Number – a unique integer from 1 to 31 which is associated with a password and specifies read or read/write access for data sets.

Path – the association, by the key item value, which links a detail data set to a master data set.

Path Count – a number from 0 to 16 which is found in the schema and specifies how many items in detail sets a master has paths to.

Read List – a list of password numbers separated by commas; it is found in the schema and specifies read-only access to a set.

Read/Write List – a list of password numbers separated by commas; it is found in the schema and specifies read/write access to a set.

Record – see Data Entry.

Root File – the file which contains all the structural information about a data base.

Schema – the means by which a data base is described to the system.

Schema Instruction – optional keywords in a schema used to provide page control and select options.

Schema Processor – a BASIC program which processes the schema text file and produces the root file.

Search Item – see Key Item.

Serial Access – a method of retrieving data entries in which all records after the current one are examined sequentially until the next non-empty one is found.

Set – see Data Set.

Stand-alone Set – a manual master or detail set that is not linked to any set.

Status Array – an integer array with at least 10 elements into which status information is returned after most data base operations.

Volume Label – a string expression of up to eight characters which is printed on a disc with PRINT LABEL.

Volume Specifier – a string expression containing the volume label preceded by a comma.

Appendix B
Statement Summary

DBBACKUP data base name [; maintenance word] [, set list] TO file specifier; return variable, return string
DBBACKUP return variable, return string
The first syntax sets up the desired backup operation. Use the second syntax for all subsequent DBBACKUPs until the backup is complete.

Possible values of the return variable are –

1. Backup completed normally.
2. The data set in the return string is about to be backed up. Do a secondary DBBACKUP using the return string.

3-4. Not Used.

5. Another backup destination file is needed. Put its file specifier into the return string and do a secondary DBBACKUP, but do not insert the medium.
6. Insert the medium indicated when the return variable was 5 and do a secondary DBBACKUP. Don't alter the return string.
7. Insert the source volume indicated in the return string, execute a VOLUME DEVICES ARE for it and do a secondary DBBACKUP.

DBCLOSE (data base name, variable, mode, status array)
Closes the data base or writes all buffered information to a disc. The variable parameter can be any expression; it is ignored. The possible mode parameters are –

1 Simple close
4 Write buffered information to disc without closing

DBCREATE data base name [; maintenance word] [, set list | * | msus | volume specifier]
Creates the data set files on the volume specified in the schema. The final parameter, if included, has the following meaning –

set list – creates only the sets specified
* – a string expression containing an asterisk. It creates all sets not yet created.
msus – creates all sets designated to reside on that medium
volume specifier – creates all sets designated to reside on that volume

DBDELETE (data base name, set, 1, status array)
Deletes the current record of the specified set (the last entry involved in a DBGET or DBPUT). The current record pointer is unchanged.

DBERASE data base name [; maintenance word][, set list / * / msus / volume specifier]

Erases all data entries and related path information from one or more of the sets in a data base. The final parameter, if included, has the following meaning –

set list – erases only the sets specified

* – A string expression containing an asterisk. It erases all sets that are created

msus – erases all sets on the specified medium

volume specifier – erases all sets on the specified volume

DBFIND (data base name, set, 1, status array, item, argument)

Used with detail sets to locate and set the current record pointer to the first entry in the chain which has the key item value identified by the argument parameter. This enables a chained DBGET to be performed.

The item parameter can be a string expression containing the desired key item name or a numeric expression containing the desired key item number. The argument parameter is a numeric or string expression containing the desired key item value.

DBGET (data base name, set, mode, status array, @, buffer string, argument)

Reads an entry from the specified set and puts it into the data base buffer string. The possible values for the mode and argument parameters are –

mode	access	argument
2	Serial	ignored
4	Directed	record number
5	Chained	ignored
7	Calculated	key item value

DBINFO (data base name, qualifier, mode, status array, buffer string)

Obtains information about data items, data sets, data paths and volumes. The table below describes the possible mode values, their purposes and what the qualifier parameter is.

Mode	Purpose	Qualifier
101	Returns item number	data item name or number
102	Describes a data item	data item name or number
104	Identifies all data items in a set	data set name or number
201	Identifies data set number	data set name or number
202	Describes a data set	data set name or number
203	Identifies all sets in a given data base	ignored
204	Identifies all data sets containing an item	data item name or number
301	Identifies data paths for a set	data set name or number
302	Identifies a key item for a set	data set name or number
401	Identifies volume number for a set	data set name or number
402	Returns a volume name	volume number
403	Identifies all the volumes in a given data base	ignored
404	Identifies all data sets on a given volume	volume name or number
501	Returns item length and position in a given set	numeric expression=item #*128+set #

DBOPEN (data base name, password, mode, status array)

Initiates access to the specified data base. The possible mode values and their meanings are –

3	exclusive read/write access with immediate posting
8	read-only access
11	exclusive read/write access with buffering

DBPURGE data base name [; maintenance word][, set list / ; * / , msus / ; volume specifier]

Purges the root file and all or some of the data files of the specified data base. The final parameter, if included, has the following meaning –

set list – purges only the sets specified

* – A string expression containing an asterisk. It purges those sets not yet purged plus the root file

msus – purges the sets on the specified medium

volume specifier – purges the sets on the specified volume

DBPUT (data base name, set, 1, status array, @, buffer string)

Adds an entry to the specified manual master or detail data set.

DBRECOVER backup file [ON msus | ON volume]; return variable, return string
DBRECOVER return variable, return string

The first syntax sets up the desired recovery operation. Use the second syntax for subsequent DBRECOVERs until the recovery is complete.

Possible values for the return variable are –

1. Recovery completed normally.
2. The data in the return string is ready to be recovered. Do a secondary DBRECOVER using the return string.
3. Not used.
4. Incomplete backup data. Ensure that backup files are used in the same order as they were created. Execute a secondary DBRECOVER.
5. Not used.
6. A new source backup file is needed. Its name is in the return string. Add the msus for the backup file to the name in the return string if it is not on the current default mass storage device, and do a secondary DBRECOVER.
7. The destination in the return string is needed. Execute a VOLUME DEVICES ARE, then a secondary DBRECOVER.

DBUPDATE (data base name, set, 1, status array, @, buffer string)

Updates the current record for the specified set (the last entry involved in a DBGET or DBPUT).

DEL FN subprogram name [TO END]
Deletes a function subprogram from memory. If TO END is specified, all subprograms following the specified one, including subroutine subprograms, are also deleted.

DEL SUB subprogram name [TO END]
Deletes a subroutine subprogram from memory. If TO END is specified, all subprograms following the specified one, including function subprograms, are also deleted.

DEV$
Returns a string whose value is the current default mass storage device.

HOLE
Returns the largest number of contiguous free records on the current default mass storage device.

IOR (numeric expression, numeric expression)
Performs a bit-by-bit inclusive-or operation on two integer expressions.

LOAD SUB file specifier [, starting line number [, increment]] [; starting subprogram [, ending subprogram]]
Loads one or more subprograms into memory from the specified program file, appending them to the current program. The lines are renumbered, if necessary. The starting line number renumbers the subprograms so they start with that number. The last line number +10 is default.

The increment defaults to 10. The starting subprogram indicates which subprogram in the file with which to begin retrieval; 1 is default. The ending subprogram indicates the last subprogram to be loaded; the last one is the default.

MSI msus
Abbreviation for MASS STORAGE IS. Specifies the current default mass storage device.

PACK USING line identifier; buffer string
Transfers data from program variables into the data base buffer string according to the format of the corresponding PACKFMT statement. It can be used before DBPUT or DBUPDATE.

PACKFMT pack list
Specifies a format to be used by PACK and UNPACK on the buffer string. The pack list can contain variables, substrings, array identifiers and skip fields separated by commas. A skip field is an X preceded by an integer for skipping character positions in the buffer string.

PRINT LABEL label [ON msus]
Writes a label onto the disc in either the current default mass storage device or the device specified after ON. The label is a string expression of 1 to 8 characters excluding any colon, comma, space, NUL(CHR$(0)) or DEL(CHR$(127)).

READ LABEL string variable [ON msus]
READ LABEL string array identifier

Reads the label off the current default mass storage device or off the device specified after ON. The second form fills a one-dimensional string array with the contents of the volume device table.

UNPACK USING line identifier; buffer string

Transfers data from the buffer string to variables in the pack list. It can be used after DBGET or DBINFO.

VOLUME DEVICE ARE msus list

Sets up the volume device table so it contains the specified msus's and their associated labels. These devices can then be accessed by IMAGE statements. The msus list is a string expression containing up to ten mass storage unit specifiers.

Appendix C

Tables, Formulas and QUERY/45 Data Types

ASCII Table

ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec	ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec	ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec	ASCII Char.	EQUIVALENT FORMS Binary	Octal	Dec
NUL	00000000	000	0	space	00100000	040	32	@	01000000	100	64	`	01100000	140	96
SOH	00000001	001	1	!	00100001	041	33	A	01000001	101	65	a	01100001	141	97
STX	00000010	002	2	"	00100010	042	34	B	01000010	102	66	b	01100010	142	98
ETX	00000011	003	3	#	00100011	043	35	C	01000011	103	67	c	01100011	143	99
EOT	00000100	004	4	$	00100100	044	36	D	01000100	104	68	d	01100100	144	100
ENQ	00000101	005	5	%	00100101	045	37	E	01000101	105	69	e	01100101	145	101
ACK	00000110	006	6	&	00100110	046	38	F	01000110	106	70	f	01100110	146	102
BEL	00000111	007	7	'	00100111	047	39	G	01000111	107	71	g	01100111	147	103
BS	00001000	010	8	(	00101000	050	40	H	01001000	110	72	h	01101000	150	104
HT	00001001	011	9	)	00101001	051	41	I	01001001	111	73	i	01101001	151	105
LF	00001010	012	10	*	00101010	052	42	J	01001010	112	74	j	01101010	152	106
VT	00001011	013	11	+	00101011	053	43	K	01001011	113	75	k	01101011	153	107
FF	00001100	014	12	,	00101100	054	44	L	01001100	114	76	l	01101100	154	108
CR	00001101	015	13	-	00101101	055	45	M	01001101	115	77	m	01101101	155	109
SO	00001110	016	14	.	00101110	056	46	N	01001110	116	78	n	01101110	156	110
SI	00001111	017	15	/	00101111	057	47	O	01001111	117	79	o	01101111	157	111
DLE	00010000	020	16	ø	00110000	060	48	P	01010000	120	80	p	01110000	160	112
DC_1	00010001	021	17	1	00110001	061	49	Q	01010001	121	61	q	01110001	161	113
DC_2	00010010	022	18	2	00110010	062	50	R	01010010	122	82	r	01110010	162	114
DC_3	00010011	023	19	3	00110011	063	51	S	01010011	123	83	s	01110011	163	115
DC_4	00010100	024	20	4	00110100	064	52	T	01010100	124	84	t	01110100	164	116
NAK	00010101	025	21	5	00110101	065	53	U	01010101	125	65	u	01110101	165	117
SYN	00010110	026	22	6	00110110	066	54	V	01010110	126	86	v	01110110	166	118
ETB	00010111	027	23	7	00110111	067	55	W	01010111	127	87	w	01110111	167	119
CAN	00011000	030	24	8	00111000	070	56	X	01011000	130	88	x	01111000	170	120
EM	00011001	031	25	9	00111001	071	57	Y	01011001	131	89	y	01111001	171	121
SUB	00011010	032	26	:	00111010	072	58	Z	01011010	132	90	z	01111010	172	122
ESC	00011011	033	27	;	00111011	073	59	[	01011011	133	91	{	01111011	173	123
FS	00011100	034	28	<	00111100	074	60	\	01011100	134	92	|	01111100	174	124
GS	00011101	035	29	=	00111101	075	61	]	01011101	135	93	}	01111101	175	125
RS	00011110	036	30	>	00111110	076	62	^	01011110	136	94	~	01111110	176	126
US	00011111	037	31	?	00111111	077	63	_	01011111	137	95	DEL	01111111	177	127

Prime Number Table

Here are the prime numbers up to 1259. The following program can be used to generate all prime numbers up to 32 767.

2	67	151	241	349	449	569	661	787	907	1021	1129
3	71	157	251	353	457	571	673	797	911	1031	1151
5	73	163	257	359	461	577	677	809	919	1033	1153
7	79	167	263	367	463	587	683	811	929	1039	1163
11	83	173	269	373	467	593	691	821	937	1049	1171
13	89	179	271	379	479	599	701	823	941	1051	1181
17	97	181	277	383	487	601	709	827	947	1061	1187
19	101	191	281	389	491	607	719	829	953	1063	1193
23	103	193	283	397	499	613	727	839	967	1069	1201
29	107	197	293	401	503	617	733	853	971	1087	1213
31	109	199	307	409	509	619	739	857	977	1091	1217
37	113	211	311	419	521	631	743	859	983	1093	1223
41	127	223	313	421	523	641	751	863	991	1097	1229
43	131	227	317	431	541	643	757	877	997	1103	1231
47	137	229	331	433	547	647	761	881	1009	1109	1237
53	139	233	337	439	557	653	769	883	1013	1117	1249
59	149	239	347	443	563	659	773	887	1019	1123	1259
61											

```
10    ! ********************************************************************
20    ! This program enables you to generate all the prime numbers up
30    ! to 32 767 for use in determining master data set capacities.
40    ! It uses a PRIME SIEVE routine.
50    ! ********************************************************************
60    !
70    DIM X$[32767]
80    INPUT "ENTER AN INTEGER BETWEEN 1 AND 32767",N
90    IF (N<1) OR (N>32767) THEN GOTO 80          ! Range checking
100   X$=RPT$("1",N)                              ! Fill X$ with 1's
110   K=2                                         ! Initialize K
120 Start_loop:    IF X$[K;1]<>"1" THEN Increment ! Loop contains SIEVE
130                PRINT K;                       ! Print a prime number
140                M=N DIV K          ! Calculate # of integers divisible by K
150                FOR J=1 TO M
160                   X$[J*K;1]="0"               ! Eliminate multiples of K
170                NEXT J
180 Increment:     K=K+1
190                IF K*K<=N THEN Start_loop      ! Rest of numbers are prime
200 Print_primes:  IF X$[K;1]="1" THEN PRINT K;
210                K=K+1
220                IF K<=N THEN Print_primes
230   END
```

Determing File Sizes

The following formulas can be used to determine the sizes of various files.

Root File (in bytes)

1600+20 per item+(40+4 per item+4 per path) for each set

Data Set Files (in bytes)

For master sets: Capacity * (sum of item lengths+6+6 per path)
For detail sets: Capacity * (sum of item length+4 per path)
For stand-alone details: Capacity * (sum of item lengths+4)

Using QUERY/45 Data Types

QUERY/45 has three special data types, **Date**, **Code** and **Name**, which provide ease of entry and checking for data validity. Each of the three special types is interpreted and converted by QUERY/45 by means of special algorithms. They were introduced briefly in Chapter 2 of this manual and are covered in detail in Chapter 3 of the QUERY/45 User's Guide.

In order for IMAGE/45 programs to use a data base containing these special data types, certain guidelines and algorithms must be used to store and retrieve data item values. This information is provided in this section.

Using the Name Type

QUERY/45 allows personal names to be entered in various ways, but always stores them in a standard format. An IMAGE/45 program should store and retrieve Name data items according to this format. The format for the Name data type is –

Last name, first name [middle name] [, title]

For example, "John H. Smith, III" would be stored as "Smith,John H.,III".

Using the Code Type

The Code data item type allows up to 35 15-character string values be specified for a data item. Each of these values is then assigned a unique integer between 1 and the number of values specified. When a Code type value is stored, the integer is stored rather than the actual value. QUERY/45 automatically performs the conversion to and from the actual value. If a data entry does not contain a value for a Code type data item, QUERY/45 enters 0.

An IMAGE/45 program should store and retrieve the integer rather than the actual string value. Therefore, it is necessary to obtain a list of the allowable values and their corresponding integers to use when writing an IMAGE/45 program. This list can be obtained in either of two ways.

The first way to obtain the list of allowable values is by using the SHOW SCHEMA feature of QUERY/45. This is covered in Chapter 4 of the QUERY/45 User's Guide.

A second way to access the list of allowable values is for your program to access the **Information File** which QUERY/45 maintains for each data base. The Information File keeps information about a data base which is unique to QUERY/45, such as synonyms for data names and allowable code values. The Information File is covered in Appendix B of the QUERY/45 User's Guide.

Using the Date Type

Data items of the Date type enable QUERY/45 users to enter a date in any one of the following ways. The interpretation of the month-day order is specified at the beginning of QUERY/45.

1-2-80	2-1-80	1-2-1980	2-1-1980	January 2, 80	January 2, 1980
1/2/80	2/1/80	1/2/1980	2/1/1980	Jan 2, 80	Jan 2, 1980
1 2 80	2 1 80	1 2 1980	2 1 1980	2 Jan 80	2 Jan 1980
1.2.80	2.1.80	1.2.1980	2.1.1980	2 January 80	2 January 1980

A Date-Verification algorithm takes a date in any of these formats and provides integer values for the month, day and year. These three integers are then used by a Date-Encoding algorithm, producing a unique integer between −999998 and 999999. If a data entry does not contain a value for a Date type data item, QUERY/45 enters −999999.

To use the Date data type, and IMAGE/45 should used the Date-Decoding and Date-Encoding algorithms. If desired, the Date-Verification algorithm can also be used. An example of using a Date type can be found with the DBPUT statement in Chapter 4 of this manual.

Following are the listings of the three date-handling routines. They have many lines of comments which serve as documentation. You may use any of the lines as are necessary for your application.

```
10    ! ************************************************************************
20    !
30    ! VERIFY DATE   Function
40    !
50    !    ACTION:  Verify date inputs month, day, and year to determine if this
60    !             date is a valid calendar date. The value '1' is returned if
70    !             the date is valid; otherwise, the value '0' is returned.
80    !
90    !
100   !    PARAMETERS:
110   !
120   !       M          - Integer (1-12) for month
130   !       D          - Integer (1-31) for day
140   !       Y          - Integer (0-5475) for year
150   !
160   !
170   !    VARIABLES:
180   !

190   !       D          - Day
200   !       Last_day   - Last valid day of the specified month
210   !       M          - Month
220   !       Y          - Year
```


```
230   !
240   !
250   !
260   !    POSSIBLE ERRORS:
270   !
280   !       No errors are expected to occur. If any parameter is not an
290   !       integer or not in the valid range, the return value is '0'.
300   !
310   !
320   ! ************************************************************************
330   DEF FNVerify_date(INTEGER M,D,Y)
340   !
350           INTEGER Last_day
360   !
370   ! *****
380   ! **
390   !       Test month, day, and year for valid ranges.
400   ! **
410   ! *****
420           IF (D<1) OR (Y<0) OR (Y>5475) OR (M<1) OR (M>12) THEN RETURN 0
430           IF (Y=5475) AND ((M>9) OR (M=9) AND (D>11)) THEN RETURN 0
440   !
450   ! *****
460   ! **
470   !       If the month is January, March, May, July, August, October, or
480   !       December, go to D31 where the last day of the month is set to 31.
490   !       If the month is April, June, September, or November, go to D30
500   !       where the last day of the month is set to 30. If the month is
510   !       February go to D28 to determine if the year is a leap year.
520   ! **
530   ! *****
540           ON M GOTO D31,D28,D31,D30,D31,D30,D31,D31,D30,D31,D30,D31
550   !
560   ! *****
570   ! **
580   !       January, March, May, July, August, October, and December have
590   !       31 days.
600   ! **
610   ! *****
620 D31:      Last_day=31
630           GOTO Got_da
640   !
650   ! *****
660   ! **
670   !       April, June, September, and November have 30 days.
680   ! **
690   ! *****
700 D30:      Last_day=30
710           GOTO Got_da
720   !
730   ! *****
740   ! **
750   !       If the year is a leap year, February has 29 days; otherwise,
760   !       it has 28 days.
770   ! **
780   ! *****
790 D28:      Last_day=28
800   !
810   ! *****
820   ! **
830   !       If a year is not divisible by 4, it is not a leap year.
840   ! **
850   ! *****
```

```
860           IF Y MOD 4 THEN Got_da
870           Last_day=29
880   !
890   ! *****
900   ! **
910   !       If a year is divisible by 4, it is a leap year unless it is
920   !       divisible by 100.
930   ! **
940   ! *****
950           IF Y MOD 100 THEN Got_da
960           Last_day=28
970   !
980   ! *****
990   ! **
1000  !       If a year is divisible by 100 but not divisible by 400,
1010  !       it is not a leap year.
1020  ! **
1030  ! *****
1040          IF Y MOD 400 THEN Cont
1050          Last_day=29
1060  !
1070  ! *****
1080  ! **
1090  !       If a year is divisible by 400 or by 1000, but not divisible
1100  !       by 4000, it is a leap year.
1110  ! **
1120  ! *****
1130 Cont:    IF Y MOD 1000 THEN Got_da
1140          Last_day=29
1150  !
1160  ! *****
1170  ! **
1180  !       If a year is divisible by 4000, it is not a leap year.
1190  ! **
1200  ! *****
1210          IF Y MOD 4000 THEN Got_da
1220          Last_day=28
1230  !
1240  ! *****
1250  ! **
1260  !       If the day is less than or equal to the last day of the specified
1270  !       month, a '1' is returned; otherwise a '0' is returned.
1280  ! **
1290  ! *****
1300 Got_da:  RETURN D<=Last_day
1310          FNEND
1350  ! ******************************************************************************
*
1360  !
1370  ! ENCODE DATE   Function
1380  !
1390  !    ACTION:  Encode date receives numbers for month, day, and year,
1400  !             and calculates the unique integer between -999998 and
1410  !             999999 corresponding to this date. If the month-day-year
1420  !             combination do not represent a valid calendar date in the
1430  !             legal date range January 1, 0000 through September 11, 5475
1440  !             the null value -999999 will be returned.
1450  !
1460  !
1470  !    PARAMETERS:
1480  !
1490  !       D          - Day
1500  !       Dcode      - Return value containing encoded date
```

```
1510  !       M          - Month
1520  !       Y          - Year
1530  !
1540  !
1550  !    VARIABLES:    (see parameters)
1560  !
1570  !
1580  !    POSSIBLE ERRORS:
1590  !
1600  !            The function Verify_date is called to verify that the month,
1610  !            day, and year values describe a valid calendar date. If this
1620  !            function returns a '0' indicating that the date is illegal or
1630  !            not in the allowed range Jan. 1, 0000 through Sept. 11, 5475
1640  !            the date code is set to the null value -999999 and returned.
1650  !
1660  ! ************************************************************************
1670  DEF FNEncode_date(INTEGER M,D,Y)
1680          SHORT Dcode
1690          IF NOT FNVerify_date(M,D,Y) THEN RETURN -999999
1700          Real=INT(365.25*Y+.75)+INT(30.55*M-29.95)-2*(M>2)+D+((M>2) AND (Y/
4=INT(Y/4)))-((M>2) AND (Y/100=INT(Y/100)))
1710          Dcode=Real+((M>2) AND ((Y/400=INT(Y/400)) OR (Y/1000=INT(Y/1000)))
)-((M>2) AND (Y/4000=INT(Y/4000)))-999999
1720          RETURN Dcode
1730          FNEND
1740  !
1750  ! ************************************************************************
*
1760  !
1770  ! DECODE DATE   Subroutine
1780  !
1790  !    ACTION:  Decode date receives an encoded date value from -999998 to
1800  !             999999 and determines the month, day, and year represented
1810  !             by this code. If the encoded value is -999999 (the null
1820  !             value) then month, day, and year will be set to 0.
1830  !
1840  !
1850  !    PARAMETERS:
1860  !
1870  !       D          - Day
1880  !       Dcode      - Encoded date
1890  !       M          - Month
1900  !       Y          - Year
1910  !
1920  !
1930  !    VARIABLES:
1940  !
1950  !       D          - Day
1960  !       D_code     - Intermediate value
1970  !       Dcode      - Encoded date
1980  !       M          - Month
1990  !       Mflag      - '1' if the year is a leap year; '0' otherwise
2000  !       Y          - Year
2010  !
2020  !
2030  !    POSSIBLE ERRORS:
2040  !
2050  !            No errors are expected to occur. If the date code is not in
2060  !            the valid range, then month, day, and year are set to zeros.
2070  !
2080  ! ************************************************************************
```

```
2090  SUB Decode_date(SHORT Dcode,INTEGER M,D,Y)
2100      IF (Dcode)-999999) AND (Dcode<=999999) THEN Valid
2110      M=D=Y=0
2120      SUBEXIT
2130 Valid: D_code=Dcode+999999
2140      Y=INT((D_code-1)/365.25)
2150      D_code=D_code-INT(365.25*Y+.75)
2160      M=INT((D_code+31)/30)
2170      Mflag=(M>2) AND (Y/4=INT(Y/4))-(Y/100=INT(Y/100))+((Y/400=INT(Y/400))
OR (Y/1000=INT(Y/1000)))-(Y/4000=INT(Y/4000))
2180      IF INT(30.55*M-29.95)-2*(M>2)+Mflag>=D_code THEN M=M-1
2190      D=D_code-INT(30.55*M-29.95)+(2-Mflag)*(M>2)
2200      SUBEXIT
2210      SUBEND
```

Appendix D
Error Messages

1	Missing ROM or configuration error. Also, check to see if all option ROMs are installed properly.
2	Memory overflow; subprogram larger than block of memory. Also check to see if your arrays are too large to fit in memory.
3	Line not found or not in current program segment. Check the spelling of line labels and line identifiers.
4	Improper return. Branched into the middle of a subroutine.
5	Abnormal program termination; no END or STOP statement.
6	Improper FOR/NEXT matching.
7	Undefined function or subroutine. Check spellings.
8	Improper parameter matching. Check the parameter lists in SUB and CALL, and DEF FN and FN statements to see if they match in number and type.
9	Improper number of parameters. Check the number of arguments used in an FN or CALL reference.
10	String value required.
11	Numeric value required.
12	Attempt to redeclare variable. Once a variable name has been declared in a DIM, COM, REAL, SHORT or INTEGER statement, it can't be redeclared in that program segment.
13	Array dimensions not specified. You must dimension the array, either explicitly or implicitly.
14	Multiple OPTION BASE statements or OPTION BASE statement preceded by variable declarative statements.
15	Invalid bounds on array dimension or string length in DIM, COM, REAL, SHORT or INTEGER statement. Strings can't be longer than 32 767 characters. The range of array subscripts is −32 767 through 32 767.
16	Dimensions are improper or inconsistent; more than 32 767 elements in an array. Check for wrong number of subscripts in an array reference. Check any matrix multiplication for proper sizes.
17	Subscript out of range.

18	Substring out of range or string too long. Check substring specifiers against length of string.
19	Improper value. Check numbers being entered, especially their exponents.
20	Integer precision overflow. The range is −32 768 through 32 767.
21	Short precision overflow. Short-precision numbers have six significant digits and an exponent in the range −63 through 63.
22	Real precision overflow. Full-precision numbers have twelve significant digits and an exponent in the range −99 through 99.
23	Intermediate result overflow.
24	TAN (n*π/2), when n is odd.
25	Magnitude of argument of ASN or ACS is greater than 1.
26	Zero to negative power.
27	Negative base to non-integer power.
28	LOG or LGT of negative number.
29	LOG or LGT of zero.
30	SQR of negative number.
31	Division by zero; or X MOD Y with Y = 0.
32	String does not represent valid number or string response when numeric data required. Check any use of VAL function and its argument. Check for correct spelling of variable name.
33	Improper argument for NUM, CHR$, or RPT$ function.
34	Referenced line is not IMAGE statement. Check the line identifier in the PRINT USING statement.
35	Improper format string.
36	Out of DATA. Make sure READ and DATA statements correspond. Use RESTORE if appropriate.
37	EDIT string longer than 160 characters. Try using a substring.
38	I/O function not allowed. TYP and other I/O functions aren't allowed in any I/O statement like DISP or PRINT. Place the value into a variable.
39	Function subprogram not allowed. An FN reference isn't allowed in any I/O statement, or in redim subscripts. Place the value into a variable.
40	Improper replace, delete or REN command. SUB and DEF FN can only be replaced by another SUB or DEF FN. They can only be deleted if the rest of the corresponding subprogram is deleted. A renumbering may cause out-of-range line numbers if completed, so an error occurs; check increment value.
41	First line number greater than second.
42	Attempt to replace or delete a busy line or subprogram. Typically, this is caused by trying to delete an input statement that is still requesting values.

43 Matrix not square. The dimensions of an identity matrix or of one used to find an inverse or determinant must be the same size.

44 Illegal operand in matrix transpose or matrix multiply. The result matrix can't be one of the operands.

45 Nested keyboard entry statements.

46 No binary in memory for STORE BIN or no program in memory for SAVE. Check line numbers in SAVE against program in memory.

47 Subprogram COM declaration is not consistent with main program. Check number, type and dimensions of variables.

48 Recursion in single-line DEF FN function. Only subprograms can be called recursively.

49 Line specified in ON declaration not found.

50 File number less than 1 or greater than 10.

51 File not currently assigned. Execute an ASSIGN statement for the file, or check the accuracy of the file number used.

52 Improper mass storage unit specifier. Check the values of the select code, unit code and controller address.

53 Improper file name. A file name can have 1-6 characters and can't contain a colon, quote mark, NULL or CHR$(255).

54 Duplicate file name. Choose another name or PURGE the old one.

55 Directory overflow. There is a maximum number of files that a mass storage medium can hold. A file will have to be removed to add another.

56 File name is undefined. Check the spelling.

57 Mass Storage ROM is missing. Check to see that the ROM is installed properly.

58 Improper file type. Use LOAD for PROG files, ASSIGN and GET on DATA files and LOADKEY for KEYS files.

59 Physical or logical end-of-file found. Attempting to READ# or PRINT# past the end of the file. Compare the data list to the file size.

60 Physical or logical end-of-record found in random mode. Compare the data list to the record size.

61 Defined record size is too small for data item. You can either PURGE and RE-CREATE the file with longer records or regroup the data being recorded.

62 File is protected or wrong protect code specified. Check to see that the protect code is included and spelled properly.

63 The number of physical records is greater than 32 767. That's the limit; use something smaller.

64 Medium overflow (out of user storage space). A file can't be set up because there isn't enough space. Use another medium or purge unwanted files.

65	Incorrect data type. You can't use GET on a DATA file that doesn't contain a program. Use TYP to find out what kind of data the computer is trying to be read.
66	Excessive rejected tracks during a mass storage initialization. The medium can't be initialized. If the medium is a flexible disk, use a different one. If the medium is a hard disc, call your HP Sales and Service Office for assistance, to determine whether there has been a hardware failure.
67	Mass storage parameter less than or equal to 0. Check values of variables. Record numbers, record lengths and number of defined records must be positive numbers.
68	Invalid line number in GET or LINK operation. Check line numbers. May be trying to LINK to file that doesn't contain a program.
69	Format switch on the disc off. Turn it on.
70	Not a disc interface. Check mass storage unit specifier.
71	Disc interface power off. Turn it on.
72	Incorrect controller address, controller power off, or disc time out. Check mass storage unit specifier; make sure controller is on.
73	Incorrect device type in mass storage unit specifer.
74	Drive missing or power off.
75	Disc system error, type I[1].
76	Incorrect unit code in mass storage unit specifier.
77	Disc system error, type II[1].
78–79	Reserved for future use.
80	Cartridge out or door open. Also check to see if interface is connected properly.
81	Mass storage device failure. Possible power failure.
82	Mass storage device not present. Check mass storage unit specifier.
83	Write protected. Check the write-protection device on the medium or drive.
84	Record not found. There is a bad spot on the medium.
85	Mass storage medium is not initialized.
86	Not a compatible tape cartridge.
87	Record address error; information can't be read. Hardware failure. Check for a dirty read head.
88	Read data error. Hardware failure. Check for a dirty read head.
89	Check read error.
90	Mass storage system error.
91–99	Reserved for future use.

100	Item in print using list is string but image specifier is numeric.
101	Item in print using list is numeric but image specifier is string.
102	Numeric field specifier wider than printer width.
103	Item in print using list has no corresponding image specifier.
104	ON KBD or TOPEN not allowed in subprogram.
105–109	Reserved for future use.
110	Plotter type specification not recognized. Check spelling of "GRAPHICS", "9872A" or "INCREMENTAL".
111	Plotter has not been specified. Check select codes.
112	No graphics hardware installed in the System 45B.
113	LIMIT specifications out of range.
114	98036 card improperly configured.
115	TDISP not allowed unless peripheral keyboard active.
116	TOPEN is active on another select code.
117–149	Reserved for future use.
150	Improper select code.
151	A negative select code was specified that does not match present bus addressing.
152	Parity error.
153	Either insufficient input data to satisfy enter list, attempt to ENTER from source into source or enter count exhausted without linefeed.
154	Integer overflow, or ENTER count greater than 32 767 bytes or 16 383 words.
155	Invalid interface register number. (Can only specify 4-7.)
156	Improper expression type in READIO, WRITEIO, or STATUS list.
157	No linefeed was found to satify % ENTER image specifier, or no linefeed record delimiter was found in 512 characters of input.
158	Improper image specifier or nesting image specifiers more than 4 levels deep.
159	Numeric data was not received for numeric enter list item.
160	Repetition of input character more than 32 768 times.
161	Attempted to create CONVERT table or EOL sequence for source or destination variable which is locally defined in a subprogram.
162	Attempted to delete a nonexistent CONVERT table or EOL sequence.
163	I/O error, such as interface card not present, device timeout, interface or peripheral failure (Interface FLAG line=0.), stop key pressed or improper interface card type.
164	Transfer type specified is incorrect type for interface card.

165	A FHS or DMA transfer with no format specifies a count that exceeds th size of the variable, or an image specifier indicates more characters than will fit in the specified variable.
166	A NOFORMAT FHS or DMA type transfer does not start on an odd numbered character position, such as A$[3].
167	Interface status error, TRL Character or an EOI was received on an HP-IB Interface before ENTER list or image specification was satisfied.
168-183	Reserved for future use.
184	Improper argument for OCTAL or DECIMAL Function.
185	Break Table overflow.
186	Undefined BASIC label or subprogram name used in IBREAK statement.
187	Attempt to write into protected memory; or, attempt to execute instruction not in ICOM region.
188	Label used in an assembled location not found.
189	Doubly-defined entry point or routine.
190	Missing ICOM statement.
191	Module not found.
192	Errors in assembly.
193	Attempt to move or delete module containing an active interrupt service routine.
194	IDUMP specification too large. Resulting dump would be more than 32 768 elements.
195	Routine not found.
196	Unsatisfied externals.
197	Missing COM statement.
198	BASIC's common area does not correspond to assembly module requirements.
199	Insufficient number of BASIC COM items.
200-206	Reserved for future use.
207	Binaries not allowed in LOAD SUB file. Do LOAD, SAVE, SCRATCH A, GET and STORE on the file to get rid of binaries. However, the loaded program may not run after the binaries are removed.
208	Volume not mounted. Mount it and execute a VOLUME DEVICES ARE statement.
209	Operation not allowed on tape. Only the BKUP file used in DBBACKUP and DBRECOVER is allowed on tape.
210	Bad status array. It must be defined as integer precision with $\geq$ 10 elements. Check spelling and current size.

211	Improper data base specified or data base not open. Improper name, or performing data base operation with invalid name.
212	Data set not found. Check set name or number and make sure it is on the volume specified in the schema.
213	Reserved for future use.
214	Data base requires creation. Perform a DBCREATE.
215–217	Reserved for future use.
218	Volume name not part of data base. Check spelling.
219	Out of available memory for a DBOPEN, DBBACKUP or DBRECOVER. Out of read/write memory if executed from main program. Out of special area if executed from subprogram, so perform the DBOPEN in the main program.
220	Improper or illegal use of maintenance word. Check spelling and leading or trailing blanks.
221	Data set not created.
222	Reserved for future use.
223	Improper backup file. In DBRECOVER, backup file has incorrect information in header or no primary DBBACKUP/RECOVER currently in progress (for secondary operation).
224	Incomplete backup file. More than one volume in backup; probably mounted in the wrong order. Start the recovery over.
225	Improper utility version number in root file. Rerun Schema Processor to generate new root file.
226	Corrupt data base – must purge and redefine. Purge root file and run Schema Processor.
227	Corrupt data base – all sets require erasure. When erasing a detail data set, ensure that all related master data sets are on-line.
228	Data sets cannot be re-created without root file.
229	Operation not allowed while DBOPEN current. Perform a DBCLOSE mode 1.
230	Improper set list in DBBACKUP, DBCREATE, DBERASE, DBPURGE or duplicate sets in the set list.
231–232	Reserved for future use.
233	Required data set root file not mounted. Mount it and perform a VOLUME DEVICES ARE.
234	Referenced line not a PACKFMT statement. Make sure line identifier is correct and that it references a PACKFMT statement.
235	Reserved for future use.
236	Insufficient length in a PACK statement, or insufficient current length in an UNPACK. Insufficient length in a DBBACKUP or DBRECOVER statement.
237	List length > 32 767 in PACK or UNPACK. Array in PACKFMT too large. Make sure it is the correct variable; redimension if necessary.

238	Numeric conversion error. Improper real number found. Check PACKFMT to make sure a REAL or SHORT variable, not INTEGER is being unpacked.
239	UNPACK requires a source string of greater length.
240–329	Reserved for future use.
300	CCOM area not allocated
301	Not allowed when channel is active
302	CMODEL statement required
303	Not allowed when trace is active
304	Too many characters in CWRITE
305	New CCOM size not allowed when channel is active
306	98046 card failure
307	Insufficient CCOM allocation
308	Illegal character in CWRITE of non-TRANSPARENT data
309	Not allowed for this CMODEL
310	CCONNECT statement required
311	Not allowed while Data Comm is suspended
312	Improper CSTATUS array
313–329	Reserved for future use.
330	Lexical table size exceeds array size.
331	Improper pointer array*.
332	Non-existent dimension specified in MAT REORDER.
333	Pointer array contains out-of-range subscript value.
334	Pointer array length does not equal number of records.
335	Pointer array is not one-dimensioned.
336	Number of records (plus twice the number of secondary keys plus twice the number of substrings) exceeds 16 383.
337	Subscript extends beyond dimensioned maximum length.
338	Subscript out-of-range in key specifier.
339	Starting location is an out-of-range subscript value.
340	Lexical table is too small to include all characters.
341	Main lexical table length plus mode section length does not equal specified table length.
342	Array is not one-dimensioned or is not integer.

* This error occurs when data is lost in the process of reordering the array. If this error does not occur, it does not necessarily imply that the pointer array contains a permutation.

343 Lexical mode section pointer out-of-range.

344 Lexical table length exceeds 16 383.

System Error octal number; octal number

This error indicates a malfunction in the machine's firmware system. Contact your Sales and Service Office.

I/O Device Errors

Two error messages can occur when attempting to direct an operation to an I/O device that is not ready for use. A printer which is out of paper or no device at a specifed select code are examples. The first message that appears is –

I/O ERROR ON SELECT CODE select code

If the condition is not corrected, the machine beeps intermittently and the following message replaces the first –

I/O TIMEOUT ON SELECT CODE select code

The I/O device can be made usable by correcting the error (loading paper, or changing the select code, for example), then executing the READY# command –

READY# select code

This command readies the I/O device and the operation which was attempted is attempted again. The select code must be specified by an integer.

If you get an I/O error on select code 0 and the printer is not out of paper, call your Sales and Service Office.

In some cases, such as an interface which is not connected, READY# for that select code may not solve the I/O error. In this case, STOP should be pressed to regain control of the computer. Be sure to turn the power off before inserting an interface. After the problem is remedied, the operation or program can be tried again.

If you get an I/O error and you have an ON KBD statement in effect, you must press STOP to gain control of the computer. Otherwise, the READY# command will be trapped by ON KBD.

CSTATUS Element 0 Errors

10	Timeout before connection
11	Clear to Send line false or missing clock
100	Channel MEMLIMIT overflow
101	Illegal protocol from remote
102	Input buffer overflow
103	Internal buffer overflow
104	Autodisconnect forced
105	RETRIES count exceeded
106	NOACTIVITY timeout
200	98046 buffer overflow

Assembly-Time Errors

DD	Doubly-defined label
EN	END instruction missing; or module name does not match.
EX	Expression evaluation error.
LT	Literal pools full or out of range.
MO	ICOM region overflow.
RN	Operand out of range.
SQ	Argument declaration pseudo-instruction out of sequence.
TP	Incorrect type of operand used.
UN	Undefined symbol.
900–999	Reserved for user.

Schema Processor Errors

The following messages do not affect program execution; they are output for information only.

Count has bad format – The number following ERRORS= is not an integer from 0 through 999, or the number following LINES= is not an integer from 20 through 999 in the $CONTROL instruction.

Illegal Command – The statement is not a $PAGE, $TITLE, or $CONTROL instruction, or the parameters are not separated by commas.

Improper command parameter – The parameter specified in the $CONTROL instruction is not valid.

Missing quotation mark – The character string specified in a $PAGE or $TITLE must be delimited by quotation marks.

Error Messages

The following messages do not halt schema processor operation; however they indicate that the root file will not be created.

Auto Master must have search item only – Automatic master data sets must contain entries with only one data item which must be a key item.

Bad Capacity or terminator – Either the number in the CAPACITY: statement must be an integer between 1 and 32 767 or a semicolon is missing. Capacity that is a power of 2 (> 10) is not allowed in a master set.

Bad Character in Password number – Password number must be an integer from 1 to 31.

Bad Data Base name or terminator – The data base name in the BEGIN DATA BASE statement is not a valid data base name of 1 through 4 alphanumeric characters beginning with a letter; it must be followed by a semicolon.

Bad Data Set type – Data set type designator must be AUTOMATIC (or A), MANUAL (or M), or DETAIL (or D).

Bad Item format or delimiter – The item's control number must be an integer from 1 through 127, and be enclosed in parentheses.

Bad Item length or terminator – The string item length must be an integer from 1 through 1022.

Bad Item type designator – The item type designator must be I, S, L, or X.

Bad Password word or terminator – The password is not followed by a semicolon, or contains illegal characters.

Bad Path Count or terminator – The path count of the master set must be an integer from 0 through 16 (or 1 through 16 for automatic masters), and must be enclosed in parentheses.

Bad Path specifier delimiter – No path specifier is present, or is not enclosed in parentheses.

Bad Read password or terminator – The read-only password number must be an integer from 0 through 31, and followed by a comma or slash.

Bad read/write specification delimiter – The read/write specifiers are not enclosed in parentheses.

Bad Set label or terminator – The set's volume label cannot be null or longer than eight characters.

Bad Set name or terminator – The data set name must be 1 through 15 alphanumeric characters beginning with a letter, and be followed by a comma.

Bad dimension or terminator – The dimension for a compound data item must be an integer from 1 through 255.

Bad terminator – ';' or ',' expected – Items within an entry definition must be followed by commas. The last item in the entry definition must be followed by a semicolon.

Bad terminator – ';' expected – The Begin Data Base line, item definition line, or NAME: line must be terminated by a semincolon.

Bad write password or terminator – The read/write password number must be an integer from 0 through 31, and be followed by a comma or a right parenthesis.

'BEGIN DATA BASE' expected – A Begin Data Base line must be the first, non-instruction line in the schema definition

'CAPACITY:' expected – A CAPACITY: line must follow the entry definition of a data set.

Data Base has no data sets – No data sets were defined in the set part of the schema. A data base must contain at least one data set.

Duplicate Item name – The same name has been used to define more than one data item in the item part of the schema.

Duplicate Item specified – The same data item name cannot be used more than once in the entry definition of a data set.

Duplicate Set name – The same name has been used to define more than one data set in the set part of the schema.

'ENTRY:' expected – Each data set defined in the schema must contain an ENTRY: line followed by the data item names in the entry. This line must follow the NAME: line.

Entry too big – The number and size of the data items defined for an entry of the data set yields a media record length greater than 1024 bytes.

Illegal characters follow terminator – Characters followed the expected terminator. The PASSWORDS:, ITEMS: and SETS: part declaration lines must each be on separate lines.

Illegal item name or terminator – The data item name must be 1 through 15 alphanumeric characters beginning with a letter and be followed by a comma.

Illegal password number – The password number must be an integer from 1 through 31.

Item length not integral words – The character count for a string item must be even.

Item length too long – The data item length cannot exceed 1022 bytes.

Master Capacity Power of 2 not allowed – If the capacity is greater than 10, it cannot be a power of 2.

Master Data Set lacks expected details – A master data set was defined with a non-zero path count, but the number of detail key items which back-referenced that master set is less than the value of the path count.

'NAME:' or 'END.' expected – The Schema Processor expected to encounter the beginning of another data set definition or the end of the schema at this point.

Password word too long – A password defined in the passwords part of the schema cannot exceed eight characters.

Referenced set not a master – Data set referenced by the detail data set key item is another detail set rather than a master data set.

Search items not similar – Master key item must be of the same type and length as the related detail data set's key item.

Search item not simple – All data items defined as a key item must have a dimension of 1; that is, they cannot be compound.

Set has no paths available – More detail data set key items have back-referenced a master data set than was specified by the master set path count.

Set too large – The product of the media record length and the set capacity exceeds 8.7767 Mbytes (that is; the space required to store the set cannot exceed 32 767 physical records).

Too many data items – No more than 255 data items may be defined in the item part of a schema.

Too many data sets – No more than 32 data sets may be defined in the set part of a schema.

Too many items specified – No more than 127 data items may be used to define a data set entry.

Too many labels specified – No more than 24 different set volume labels may be defined.

Too many paths in a data set – No more than 16 key items may be defined in a detail data set.

Undefined item referenced – The data item appearing in the data set definition was not previously defined in the item part of the schema.

Undefined set referenced – The master data set defined by a detail key item was not previously defined in the set part of the schema.

The following errors halt execution of the Schema Processor

Insufficient Space For Scratch File – The mass storage device specified for the scratch file must contain 85 physical records of contiguous file space for the scratch file. Purge any unused files and run the program again.

Max Errors – Schema Processing Terminated – The specified or default number of errors has been exceeded.

'PASSWORDS:' not found (FATAL) – A PASSWORDS: line must immediately follow the Begin Data Base statement. Make sure you are using the correct schema text file.

Schema File Errors

Duplicate Root File Name. The destination medium contains a file of that name. Schema Processor is terminated.

Unable to open Root file. The Schema Processor is terminated. The Schema Processor returns to the beginning section.

Unable to open scratch file. The Schema Processor returns to the beginning section.

Unable to open text file. The Schema Processor returns to the beginning section.

Unexpected EOF on text file. Schema Processor is terminated.

Unexpected file error (FATAL)

UNRECOGNIZED SYSTEM ERROR (FATAL)

IMAGE Status Errors

The following are possible values and meanings of the Condition Word (first element of the status array). After an error, the status array is as follows –

Element	Description
1	Condition word is non-zero
2-4	No change
5	DBOPEN mode
6	Statement identification number
7	Program line number
8	0
9	Value of the mode parameter
10	Integer-for system use only

Each statement has an identification number.

Number	Statement
401	DBOPEN
402	DBINFO
403	DBCLOSE
404	DBFIND
405	DBGET
406	DBUPDATE
407	DBPUT
408	DBDELETE

Condition Word Value	Error Description
0	Successful execution – no error
-1	Improper data base name; already have read/write access to the data base
-10	You may not open additional data bases; five are already opened
-11	Bad data base name or preceding blanks missing. Don't change the first two characters. Data base may not be open.
-14	DBPUT, DBDELETE and DBUPDATE not allowed in DBOPEN mode 8
-21	Bad password – grants access to nothing or not to that set. Check spelling. Data item, data set, or volume nonexistent or inaccessible. Check spelling and DBOPEN password. Volume references must be numeric for DBINFO.
-22	Detail data set required
-23	You lack write access to this data set
-24	DBPUT or DBUPDATE not allowed on Automatic Master. Check correctness of set reference.
-31	Improper mode in data base statement. DBGET mode 5 bad – specified data set lacks chains

-52	Item specified is not an accessible key item in the specified set. Bad @ parameter – must be "@;" or "@ " or "@".
-74	Root file name in disc directory and name in root file are different. Make sure root file not moved or renamed.
-91	Root file version not compatible with current IMAGE/45 statements. Incorrect version of Schema Processor used.
-92	Data base requires creation
-94	Data or structure information lost. Data base must be erased or redefined.
-95	Cannot DBOPEN while a DBBACKUP or DBRECOVER is going on.
11	End of file on serial DBGET; no entries following the current record.
12	Negative record number on directed DBGET. Check record number and spelling.
13	Record number greater than capacity on directed DBGET. Check record number and spelling.
15	End of chain encountered
16	The data set is full
17	No current record or the current record is empty; make sure that a current record is defined for this set. There is no chain for the key item value. There is no entry with the specified key value
18	Broken chain. Must UNLOAD the data base.
41	DBUPDATE will not alter a key item. Make sure correct key item values are in the correct places in the buffer string.
43	Duplicate key item value in master not allowed.
44	Can't delete a Master entry with non-empty detail chains
50	Buffer string is too small for requested data. Redimension if necessary.
53	Argument parameter type incompatible with key field type (DBGET, mode 7 or DBFIND) or Argument's current length of string argument is less than the string length of the key item value.
80	Data set's volume is not on line; or set not created.
94	Corrupt data base successfully opened in mode 8
1xx	There is no chain head for path xx. Add key item value to related master set.
3xx	The automatic master for path xx is full.
4xx	The master data set for path xx is not on-line (Applies to DBPUT and DBDELETE for detail data sets)
500	Root file volume isn't mounted.
5xx	Needed volume on-line; created data set xx isn't there.

Subject Index

a

b

c

d

e

f

g

h

i

k

l

m

n

o

p

q

r

s

t

u

v

Your Comments, Please...

Your comments assist us in improving the usefulness of our publications; they are an important part of the inputs used in preparing updates to the publications.

In order to write this manual, we made certain assumptions about your computer background. By completing and returning the comments card on the following page you can assist us in adjusting our assumptions and improving our manuals.

Feel free to mark more than one reply to a question and to make any additional comments.

Please do not use this form for questions about technical applications of your system or requests for additional publications. Instead, direct those inquiries or requests to your nearest HP Sales and Service Office.

If the comments card is missing, please address your comments to:

HEWLETT-PACKARD COMPANY
Desktop Computer Division
3404 East Harmony Road
Fort Collins, Colorado 80525 U.S.A.

Attn. Customer Documentation
Dept. 4231

All comments and suggestions become the property of Hewlett-Packard.

Guidelines for IMAGE/45 Data Base Operations

- Back up data bases regularly.
- The root file must always be on-line while the data base is open. Do not move the root file to a different mass storage device while the data base is open.
- Check the status array after every data base (DB) statement.
- When using DBOPEN mode 11, use DBCLOSE mode 4 frequently to provide data security.
- Before moving a data base volume while the data base is open, use DBCLOSE mode 4 to write all information to the disc. After moving a volume, execute a VOLUME DEVICES ARE statement.
- Specify a maintenance word when creating a data base to provide greater security. Protect the backup (BKUP) file after it is created to provide greater security for the backup.
- Do not use PURGE or RENAME on any data base files (ROOT, DSET or BKUP). Use the COPY statement with care.
- Do not relabel data base volumes.
- Do not press the STOP key while the data base is open. If you must do this to regain control of the machine (after an I/O error for example), be sure to execute a DBCLOSE statement from the keyboard.
- Do not put a data base on a tape cartridge.

Part No. 09845-91055
Microfiche No. 09845-98055

Printed in U.S.A.
Second Edition, Nov 1980