

IMAGE/260 Programming Manual



Herrenberger Straße 130
D-7030 Böblingen, West Germany

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change.

First Edition	Dec 1978
Update No. 1	Jun 1979
Second Edition	Nov 1979
Update No. 1	Jan 1983
Update No. 2	Jan 1984
Reprint	Jul 1984

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard GmbH.

Copyright 1979, 1981, 1983, 1984 Hewlett-Packard GmbH.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Table of Contents

Chapter 1: Introduction

What is IMAGE?	1-1
IMAGE Organization	1-2
IMAGE Structure	1-2
Data Base Organization	1-3
Types of Data Sets	1-4
Data Access	1-5
Serial Access	1-5
Directed Access	1-5
Chained Access	1-6
Calculated Access	1-7
Migrating Secondaries	1-8
Manual vs. Automatic Master Data Sets	1-9
Data Base Files	1-9
User Class Numbers and Passwords	1-10

Chapter 2: Data Base Definition

Introduction	2-1
Language Conventions	2-2
Data Base Definition Language	2-3
Data Base Name Definition	2-3
Password Definition	2-3
Item Definition	2-4
Set Definition	2-5
Master Data Set Definition	2-6
Detail Data Set Definition	2-7
Media Record Length	2-9
Schema Commands	2-9
\$TITLE	2-10
\$PAGE	2-10
\$CONTROL	2-11
Schema Processor Operation	2-13
Using the Schema Processor	2-13
Routine Messages	2-14

Chapter 3: Data Base Manipulation

Introduction	3-1
DBOPEN	3-2
DBCLOSE	3-4
DBGET	3-6
DBUPDATE	3-9
DBPUT	3-10
DBDELETE	3-12
DBFIND	3-14
DBINFO	3-16
DBLOCK	3-22
DBUNLOCK	3-25
Advanced Access Statements	3-26
DBASE IS	3-26
IN DATA SET	3-27
PREDICATE	3-30

Chapter 4: Data Base Utilities

Introduction	4-1
Data Base Utility Statements	4-2
DBCREATE	4-2
DBERASE	4-4
DBPURGE	4-6
Data Base Binary Statements	4-8
DBSTORE	4-9
DBRESTORE	4-11
Backup and Recovery	4-12
Method 1: DUPL	4-12
Method 2: DBSTORE	4-12
Data Base Utility Programs	4-13
DBUNLD	4-13
DBLOAD	4-17
Salvaging Data	4-21
Data Base Restructuring	4-22
The DBMODS Utility	4-23
The DBPASS Utility	4-31
DBPASS	4-31
DBMAINT	4-31
READ DBPASSWORD	4-32
WRITE DBPASSWORD	4-32
The XCOPY Utility	4-34

CHAPTER 1

Introduction

This manual describes the IMAGE¹ data base system software available with the HP 250. Chapter 1 gives a brief introduction to the data base concept and how IMAGE/250 is organized as a data base processing system. Chapters 2 thru 4 describe how to define, access and maintain a data base. Chapter 5 offers example data base operations and example programs.

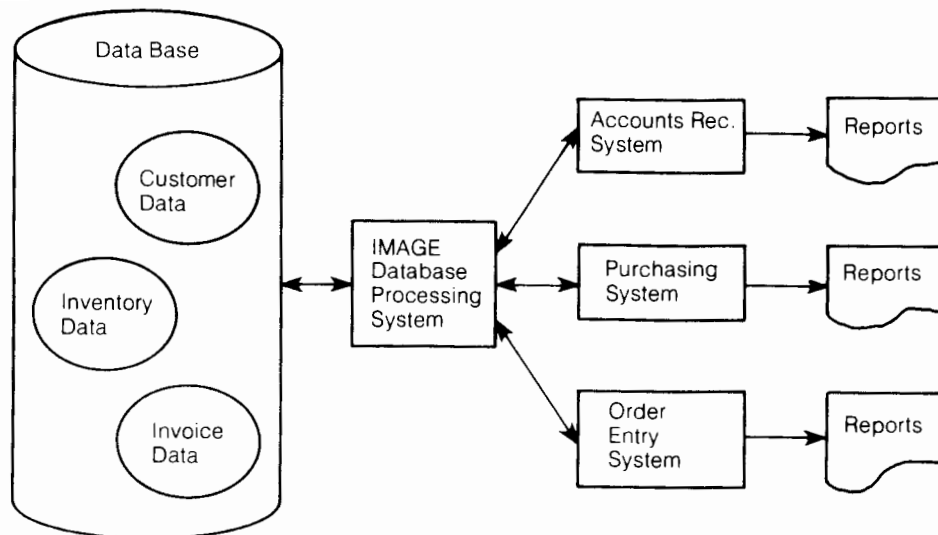
This manual assumes you are familiar with the BASIC/250 file processing operations available on the HP 250, as described in the "File Storage" chapter of the BASIC Programming Manual.

What is IMAGE?

IMAGE/250 is a set of statements and programs that operate on a data base. A **data base** is a group of logically related files which contain all the data necessary to satisfy a user's information needs. A data base also contains structural information describing how the various data files are related. Relationships that link data within a data base allow access to both related data and data across files.

An example data base system is shown in the following diagram. It performs many of the same tasks as standard file-processing systems. However, its files have been integrated into a data base that is processed by application programs. As shown in the example, accounts receivable, order entry, and purchasing systems perform their usual file-processing functions, but they call upon IMAGE/250 to access the data base. More importantly, IMAGE/250 can process the data as an integrated whole. Since the files have been created by the same system, all of the data is compatible. This allows for integrated processing, which is the ability to index across the various files to extract related information. For example, inventory data can be logically "tied to" several sales orders to represent the relationship between items in the inventory file and the sale of those items in the invoice file. This relationship can provide management with information such as the source of sales by salesman, region, customer and product type.

¹ The IMAGE/250 data base processing system is modeled after the HP IMAGE/3000 system. For a comparison of IMAGE operations between the HP 250 and the HP 3000, refer to Appendix D.



A Data Base System

IMAGE Organization

The following sections describe the structure and access methods employed by IMAGE/250.

IMAGE Structure

IMAGE/250 is organized into three sections: **data base definition**, **data base manipulation**, and **data base maintenance**.

Data base definition is accomplished using the EDITOR and SCHEMA processor programs. These programs are used in conjunction with the data base definition language (DBDL) to define the structure, size and security of a data base.

Data base access and manipulation is performed using the data base manipulation statements. These statements are invoked from BASIC language programs, and serve as an interface between data bases and applications programs. Applications programs can use the power of the BASIC/250 language for data processing, leaving the data base access and structural housekeeping activity to the manipulation statements.¹

Data base maintenance operations are performed using the data base utilities. The utilities provide the capability to create, purge, and erase data sets, and to backup and restore data bases. The utilities are also used for data base restructuring.

¹ Additional IMAGE/250 statements, available in the SORT DROM, provide the capability to select data using Boolean expressions, and to retrieve data in sorted order.

Chapter 5: Example Operations

Introduction	5-1
Data Base Design	5-1
Data Base Definition and Creation	5-7
Program Examples	5-12
Utility Examples	5-29
Backup and Recovery	5-29
Restructuring a Data Base	5-32
Data Base Locking	5-45
Lock Descriptors	5-46
Setting-up Predicates	5-48
Lock Conflicts	5-50
Defaults Without IMAGE2	5-52

Appendix A: PACK Statements

Introduction	A-1
PACKFMT	A-1
PACK	A-2
UNPACK	A-3

Appendix B: Text Editor

Introduction	B-1
Error Messages	B-2
Special Control Keys	B-3
EDITOR Commands	B-3
ADD	B-4
CHANGE	B-5
DELETE	B-6
END or EXIT	B-7
FIND	B-8
GATHER	B-9
HOLD	B-10
KEEP	B-11
LIST	B-12
MODIFY	B-13
SET	B-14
TEXT	B-15
WHILE	B-16

Appendix C: DBML Syntax

SCHEMA Definition	C-1
Set Definition Syntax	C-2
DBML Statements and Advanced Access	C-4
Utility Statements	C-7

Appendix D: IMAGE 250/3000 Comparison

IMAGE Feature Comparison	D1
IMAGE Mode Comparison	D-2

Appendix E: IMAGE/250 Error Messages

EDITOR Error Messages	E-1
IMAGE Status Errors	E-2
PACK and IMAGE/250 Error Codes	E-5
Schema Processor Messages	E-6
DBLOAD/DBUNLD Error Messages	E-13

Appendix F: HASH Algorithm Description

Introduction	F-1
PRE-OS6	F-1
Standard	F-2

Data Base Organization

There are three basic structures within an IMAGE/250 data base: data items, data entries, and data sets.

A **data item** is the smallest data element. Each data item has a value and is referenced by a data item name. Data items correspond to program variables within an applications program. Some examples are:

Data Item Name	Data Item Values
PRODUCT-NO	50 100 1000
PROD-DESC	Tricycle Standard Bicycle 10-Speed Bicycle



A **data entry**, or record, is an ordered collection of related data items. All data is transferred to and from a data base on a record basis. For example:

Data Entry Definition	PRODUCT-NO	PROD-DESC
Data Entry Values	{ 50 100 1000	Tricycle Standard Bicycle 10-Speed Bicycle

A **data set** is a collection of data entries sharing a common definition. All entries in a data set are stored as a separate file on a mass storage device, and are referred to by a data set name. Some examples are shown below:

Data Set Name: **PRODUCT**
 Set Capacity: **10 entries**
 Data Entry Definition: **PRODUCT-NO PROD-DESC**

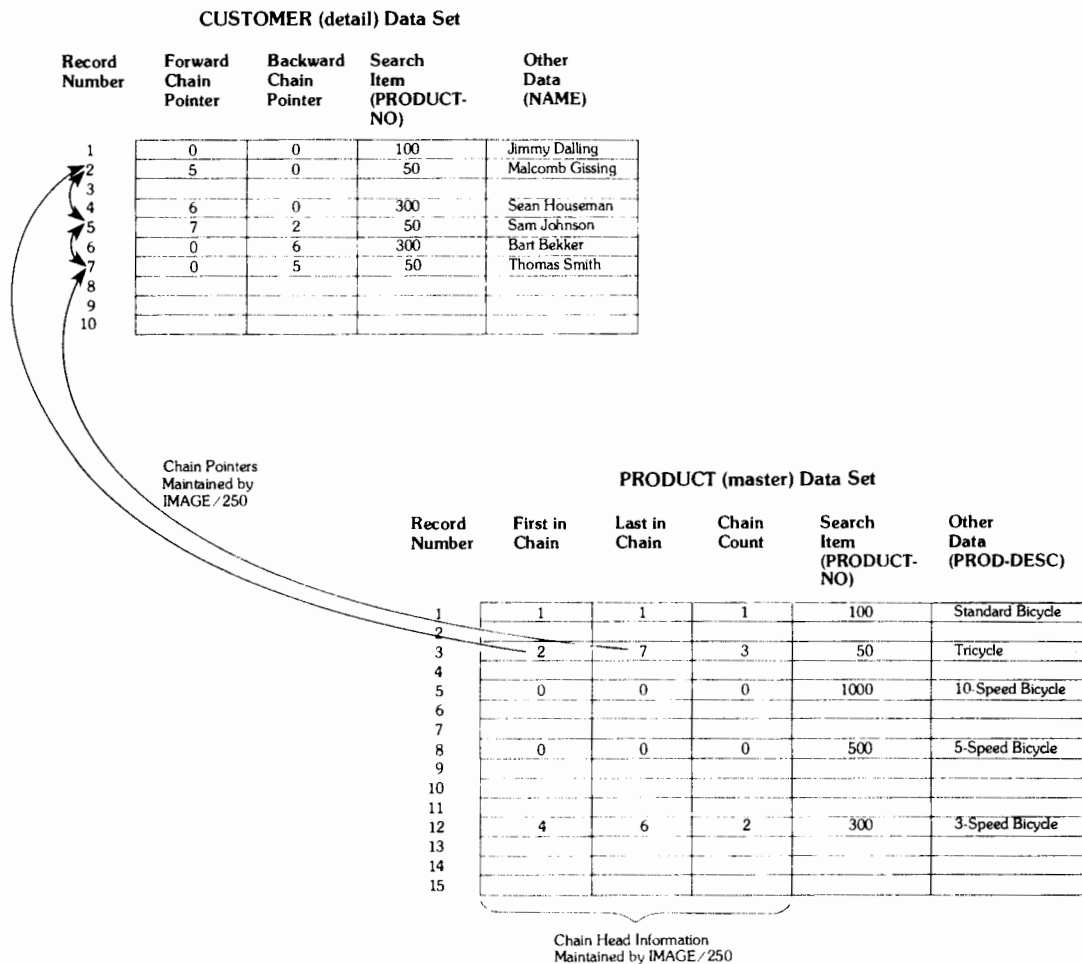
Entry or Record No.	PRODUCT-NO	PROD-DESC
1		
2	50	Tricycle
3		
4		
5		
6	1000	10-Speed Bicycle
7		
8		
9	100	Standard Bicycle
10		

Types of Data Sets

There are two types of data sets in the IMAGE/250 system: **master data sets** and **detail data sets**. Detail data sets are used to store "line item" information. Master data sets are generally used as indexes to information within detail data sets. For example, the CUSTOMER detail data set shown below contains information about a customer order, such as the customer's name and the product purchased. A related master set PRODUCT is used to point to all orders for a particular product. This association of an item in a master data set and a detail data set is known as a **data path**.

Up to eight data paths may be defined for a particular data set. The item used to link the master data set with the detail data set is known as a **search item**. Master data sets have only one search item, but detail data sets may have up to eight search items.

Data entries contain pointer information used to link related entries. Detail entries contain pointers to other entries containing the same search item value. This linkage of related detail entries is known as a **data chain**. Master entries contain pointers to the beginning and end of data chains, along with the number of entries within the chain. This chain information is automatically maintained by IMAGE/250.



Data Chain Example

Data Access

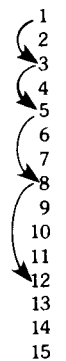
Data base access and manipulation is performed using the data base manipulation statements. These statements, which are specifically designed to interact with an IMAGE data base, are invoked through BASIC language programs. These statements are structured so that each one suggests its function (e.g., DBGET gets data from a data set). All data access is carried out at the data entry level (this is known as the “full record mode”). Data entries may be accessed in one of four modes: **serial**, **directed**, **chained** or **calculated**.

Serial Access

When accessing a data set in serial mode, IMAGE starts at the most recently accessed record (data entry), called the **current record**, and sequentially examines records until the next, non-empty record is located. This record is then transferred to the data buffer, and becomes the new current record. Serial access is often used to examine or list all entries in a data set.

The following example shows entries in the PRODUCT (master) data set. The record numbers are shown to the left of each entry. Records 1, 3, 5, 8 and 12 contain non-empty entries. The arrows to the left of the record number show how entries will be retrieved in serial mode. If the current record is 5, for example, the next record accessed in serial mode will be record number 8.

Record Number	Search Item	Other Data
1	100	Standard Bicycle
2		
3	50	Tricycle
4		
5	1000	10-Speed Bicycle
6		
7		
8	500	5-Speed Bicycle
9		
10		
11		
12	300	3-Speed Bicycle
13		
14		
15		



Serial Access of the PRODUCT (master) Data Set

Directed Access

A second method of accessing a data entry is directed access. With this method, IMAGE returns the record specified by a record number supplied by a program. If the specified record is non-empty, the record is transferred to the data buffer. If the record is empty, a status error is returned. In either case, the current record is set to the record specified. Directed access is used to read entries following a SORT or FIND operation (see the SORT / 250 Programming Manual for a description of these operations), or to read entries in reverse order along a data chain.

The following example shows the retrieval of an entry using directed access. The record number 5, supplied by an applications program, instructs IMAGE to retrieve record 5. IMAGE copies the record into the data buffer and resets the current record to 5.

Record Number	Search Item	Other Data
1	100	Standard Bicycle
2		
3	50	Tricycle
4		
5	1000	10-Speed Bicycle
6		
7		
8	500	5-Speed Bicycle
9		
10		
11		
12	300	3-Speed Bicycle
13		
14		
15		

Record Number 5 →
Supplied

Directed Access of the PRODUCT (master) Data Set

Chained Access

Chained access is used either to retrieve detail data entries with common search item values, or to retrieve master entries with synonym search item values. IMAGE/250 supports chained access in a forward direction. Entries along a data chain may be accessed in a reverse direction, however, by using directed access and the status information returned by IMAGE (see Chapter 3). Chained access of detail data sets is often used for retrieving information about related events.

The following example shows the retrieval of detail entries using chained access. The corresponding chain pointer information, maintained by IMAGE/250, is shown along with the record numbers for the set. IMAGE uses this pointer information to retrieve the next entry along the chain. The arrows to the left of the record numbers show how entries will be retrieved in chained mode. If the current record is 5, for example, the next record access in chained mode will be 7.

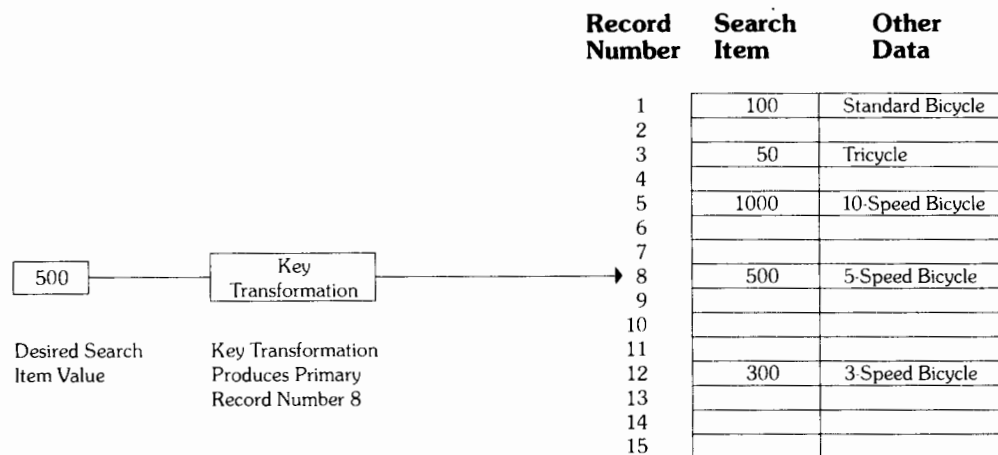
Record Number	Forward Chain Pointer	Backward Chain Pointer	Search Item (PRODUCT-NO)	Other Data (NAME)
1	0	0	100	Jimmy Dalling
2	5	0	50	Malcomb Gissing
3				
4	6	0	300	Sean Houseman
5	7	2	50	Sam Johnson
6	0	6	300	Bart Bekker
7	0	5	50	Thomas Smith
8				
9				
10				

Arrows indicate chain sequence: 1 → 2 → 4 → 5 → 7

Chained Access of the CUSTOMER (detail) Data Set

Calculated access is based on a search item value, and may be used to access master data sets only. This access method involves mapping a program-supplied search item value into a primary record number using the process of key transformation (hashing). (See Appendix F for explanation of hash algorithm.) The record at that location is then examined by IMAGE to determine if it contains the matching search item value. The key transformation may map more than one search item value into the same primary record number. When this occurs, the search item values are called synonyms. If the record at the location specified by the calculated record number does not contain the desired search item value, but does contain a synonym item value, an exhaustive search of all synonyms with the same primary record number is made to locate the desired entry. This access method eliminates the need to search every entry in the data set to locate the entry with the desired search item value. No ambiguity exists since search item values must be unique within a master data set. Calculated access is often used to retrieve selected master entries.

The following example shows the retrieval of an entry using calculated access. The search item value 500, which is supplied by an applications program, is used by IMAGE to calculate the primary record number 8. Since record number 8 contains the desired search item value, the record is copied into the data buffer and the current record is set to 8.



Calculated Access of the PRODUCT (master) Data Set

Migrating Secondaries

Migrating secondaries are an integral part of the way IMAGE handles synonyms in master data sets and must be understood to fully use the system.

When there are synonyms, the first search item is placed in the primary record. Following search items are placed in secondary records. Pointers from the primary record to the first secondary record, first secondary record to second secondary record, etc., are kept.

Search Items	Record Number	Count	Last Entry Backward	First Entry Forward	Search Item
500 } Key Transformation	8	3	10	9	500
450 }	9	0	0	10	450
25 }	10	0	9	0	25

Notice in the figure above that the primary record contains a pointer to the first and last secondary in the chain. The first secondary (record 9) points to the second secondary (record 10) but does not point back to the primary. The last secondary in the chain points back to the previous secondary, but does not point to the primary. The primary record number is automatically located via the key transformation.

If the key transformation produces a record number for a new search Item which is occupied by a secondary, the secondary search Item is moved:

Search Items	Record Number	Count	Last Entry Backward	First Entry Forward	Search Item
200 } Key Transformation	8	3	10	11	500
	9	1	0	0	200
	10	0	11	0	25
	11	0	0	10	450

If the primary search item is deleted, the first secondary is moved to replace it:

(Search item 500 deleted)	Record Number	Count	Last Entry Backward	First Entry Forward	Search Item
	8	2	10	10	450
	9	1	0	0	200
	10	0	0	0	25

When there is only one secondary search item, its pointers are set to zero.

Manual vs. Automatic Master Data Sets

A master set may be either **manual** or **automatic**. These two types of master sets have the following characteristics:

Manual	Automatic
May stand alone. Need not be related to any detail data set.	Must be related to one or more detail data sets.
May contain data items in addition to the search item.	Must contain only one data item, the search item.
Entries must be explicitly added or deleted. A related detail data entry cannot be added until a master entry with a matching search item value has been added. When the last detail entry related to a master is deleted, the master entry still remains in the data set. Before a master entry can be deleted, all related detail entries must be deleted.	IMAGE automatically adds or deletes entries when needed based on the addition or deletion of related detail data set entries. When a detail entry is added with a search item value different from all current search item values, a master entry with matching search item value is automatically added. Deletions of detail entries trigger an automatic deletion of the matching master entry if it is determined that all related data chains are empty.
The search item values of existing master entries serve as a table of legitimate search item values for all related detail data sets. Thus, a manual master can be used to prevent the entry of invalid data in the related detail data sets.	

Data Base Files

Three types of files are associated with IMAGE/250 data bases: ROOT, DSET and BKUP. These files are used to store data and data base structure information.

Each IMAGE/250 data base has one ROOT file. This file is created by the Schema Processor program (SCHEMA), and contains all of the data base structural information. A portion of the root file is copied into user memory each time the data base is opened for access. This part of the root file is called the **data base control block** (DBCBC), and contains all the dynamic information required for execution of the data base manipulation statements. When data base access is complete, the DBCBC is copied back onto the root file.

In addition to the root file, a data base may have one or more data set (DSET) files. These files are created by the data base utilities, and may reside on separate volumes. Placing data sets on separate volumes allows set capacities to be less dependent on the on-line mass-storage capacity. Only the root file and the data sets involved need be on-line during an operation. In addition, sensitive data may be stored on a separate volume and placed under lock and key to provide an additional level of security.

ROOT files and DSET files are created, accessed, and maintained only through IMAGE/250 operations. These files cannot be read, modified, copied, purged or renamed using BASIC/250 statements.

Backup (BKUP) files are created by the data base utilities, and contain a copy of a portion of a data base. Backup files are used to restore a data base following a system failure, or to load data into a restructured data base. Unlike ROOT and DSET files, BKUP files may be purged, renamed, copied and protected using BASIC/250 statements. Access to the data within these files, however, is limited to the data base utilities. Unlike other files, BKUP files may reside on more than one volume.

User Class Numbers and Passwords

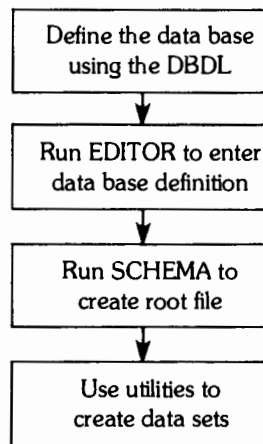
IMAGE/250 provides controlled access to data sets by allowing the definition of up to 31 user classes. Each user class is associated with a password during data base definition. When initiating access to a data base, a password must be supplied to establish a user class. The user class is then used to determine both the data sets which may be accessed and the type of access permitted (read only or read/write). The user-class capability is defined during data base definition. Security on an item level is not provided, so user classes granted access to a data set may access all items within the data set's record.

CHAPTER 2

Data Base Definition

Introduction

All IMAGE/250 data bases are defined using the data base definition language (DBDL). Once the data base has been defined using the DBDL, the HP 250 EDITOR program is used to create a data file containing the data base definition. This definition, known as a schema, is used by the Schema Processor to create the data base root file. The root file contains the structure information of the data base. The data set files of the data base are created using the Data Base Utilities. Once the root file and data set files are created, the data base is ready to be accessed by either applications programs or by QUERY/250. This data base definition sequence is shown below.



Data Base Definition Procedure

A data base definition is organized into three sections: the password part, the item part and the set part. Each section defines a particular part of the data base. Additional statements are used to specify the data base name, to select Schema Processor options, to specify page control, and to designate the end of the data base definition. A data base definition is organized as shown below:

```
BEGIN DATA BASE (data base name definition)
  PASSWORDS:
    password definition section
  ITEMS:
    item definition section
  SETS:
    set definition section
END.
```

Each data base definition statement must begin on a new line. Regardless of the actual length of each line, the Schema Processor reads and prints only the first 80 characters of each line, and processes only the first 72 characters of each line. Any remaining character positions in the line are available for comments or collating information. Comments, which are enclosed in double carrets (e.g., << comment >>), may appear at the end of any line. Any line may be continued on one additional line by following the last, non-blank character of the line to be continued with an ampersand (&). The ampersand continue character should precede any comment in that line.

Language Conventions

DOT MATRIX	Dot matrix items must appear as shown.
[]	Brackets are used to show optional items. For example, $[\begin{matrix} A \\ B \end{matrix}]$ means select A or B or neither.
{ }	Braces are used to denote required items. For example, $\left\{ \begin{matrix} A \\ B \\ C \end{matrix} \right\}$ means A or B or C must be selected.
...	An ellipsis indicates that the previous sequence may be repeated.
volume name	A 1 thru 8 character string used to specify a particular storage medium.
unit spec	A string specifying a mass storage device. The form is: <code>[:device type[select code[: device address[: unit code]]]]</code> The device types are: F Flexible Disc C HP 7906A Removeable Disc D HP 7906A Fixed Disc L Built-in Fixed Disc The select code must be equal to 2. The device address and unit code are integers from 0 thru 7. For example, the unit spec <code>:F2,6,0</code> refers to the top flexible disc drive, unit 0.
volume spec	A string expression containing a unit spec or a comma followed by a volume name.

Data Base Definition Language

Data Base Name Definition

The first part of a data base definition is the statement specifying the data base name and the root file location. The format of this statement is:

```
BEGIN DATA BASE data base name [ , volume name ] ;
```

The data base name is from 1 thru 6 characters, consisting of uppercase alphabetic characters, the numbers 0 thru 9, and the minus sign (-). The name begins with a letter. The optional volume name specifies the disc on which the root file is to be created. If the volume name is not specified, the root file will be created on the default mass storage device. This statement must be terminated by a semicolon.

For example:

```
BEGIN DATA BASE EXAMPL, FLOP ;
```

Password Definition

The password definition section follows the data base name definition. The password section begins with the statement:

```
PASSWORDS :
```

and is followed by a list of user class numbers and their corresponding passwords. Each password definition has the form:

```
user class number password ;
```

Each user class number/password pair must appear on a new line. The user class number is an integer from 1 thru 31, and must be unique within the password section. Passwords are from 1 thru 8 ASCII characters, excluding semicolons. Blanks within the password will be removed by the Schema Processor. If the same password is assigned to multiple user class numbers, the lowest-numbered class will be used. Lines containing only a user class number and a semicolon will be ignored.

For example:

```
PASSWORDS :  
31 Clerk ;  
5 Gun ball ;  
10 SECRET ;  
15 ; <<Not currently assigned >>  
22 %-+ ;
```

Item Definition

The item definition section follows the password section. The item section begins with the statement:

ITEMS:

and is followed by a list of all items to be used in the data base. Up to 255 data items may be defined in a data base. Each item definition has the form:

item name * [sub-item count] specifier [(control no.)] †

The item name is from 1 thru 15 characters, consisting of uppercase alphabetic characters, the numbers 0 thru 9, and the minus sign (-). The name begins with a letter. Item names must be unique within the item section.

The sub-item count is used to define the array length of compound data items (one dimensional item array). An omitted sub-item count, or a sub-item count of 1 specifies a simple item. The sub-item count must be an integer between 1 and the maximum value for the item type shown in the following table.

The specifier is used to declare the item type. Items may be defined to contain numeric data or ASCII string data. The string designator must be followed by the maximum string length. The string length must be even, and cannot exceed 1022 characters. The item specifiers are described below.

Item Specifier Types

Specifier	Type	Description	Range	Item' Length	Maximum Sub-item Count
L	Long (Numeric)	Denotes a 12-digit BCD floating-point number.	± 9.9999999999E+99 thru ± 1.0000000000E-99	8 bytes	127
S	Short (Numeric)	Denotes a 6-digit BCD floating-point number.	± 9.99999E+63 thru ± 1.00000E-63	4 bytes	255
I	Integer (Numeric)	Denotes a 16-bit integer number (binary).	+32767 thru -32768	2 bytes	511
X	String	Denotes an ASCII character string. Must be followed by an integer character count.	Up to 1022 characters	1 byte per character	511

† These numbers are for simple items (sub-item count equal to 1). To compute the item length for compound items, multiply the item lengths shown by the sub-item count.

The control number is used for external item formatting, and must be an integer from 0 thru 127. This number may be retrieved using an `IMAGE/250` statement, but is otherwise ignored. The control number is provided for use by applications programs. `QUERY/250`, for example, uses the number to determine the format of numeric data, and to prevent `QUERY/250` from modifying sensitive data. Refer to the `QUERY/250 Programming Manual` for more information.

An example item definition section is shown below:

```
ITEMS:
      IN-STOCK, I;
      COST, S;
      TOT-SALE, L;
      DESCRIPTION, X30;
      MONTH, 12 X10; <<12 element array>>
```

Set Definition

The set definition section follows the items section. The set section begins with the statement:

```
SETS:
```

and is followed by a list of set definitions. The statement:

```
END.
```

must follow the last set defined in the data base.

Up to 50 data sets may be defined in the set section. A data set may not span multiple volumes but different data sets may reside on separate volumes. Up to 23 unique volume labels may be used to specify the location of the data sets. A description of the statements used to define master and detail data sets follows.

Master Data Set Definition

Manual master data sets are defined using the format:

```
{NAME:} set name, {MANUAL} (read list/write list) [volume name]
{N:} M

{ENTRY:} item name, (path count),1
{E:}
    [item name ]
    [item name ]
    .
    .
    .
    [item name ]

{CAPACITY:} maximum entry count
{C:}
```

Automatic master data sets are defined using the format:

```
{NAME:} set name, {AUTOMATIC} (read list/write list) [volume name]
{N:} A

{ENTRY:} item name (path count)
{E:}

{CAPACITY:} maximum entry count
{C:}
```

The set name refers to the master set being defined. The name is from 1 thru 15 characters, and consists of uppercase alphabetic characters, the numbers 0 thru 9, and the minus sign (-). The first character must be a letter. All set names must be unique within the schema.

The read list and write list contain lists of user class numbers separated by commas, and are used to determine which user classes have access to the set. Specifying a user class number (an integer from 0 thru 31) in the read list allows that user class to have read access to the set. Specifying a user class number in the write list allows that user to have read and write access to the set. The read list may be null, but the write list must contain at least one user class number.

¹ If the entry is defined with only one item name, the ENTRY line is terminated with a semicolon instead of a comma.

The optional volume name specifies the disc on which the data set is to reside. This name is from 1 thru 8 ASCII characters. If no volume name is specified, the data set will reside on the same volume as the root file.

The item name is the name of a data item previously defined in the item definition section. Each item name must appear on a new line, and must be unique within the data set. A data entry may be defined with up to 127 item names. The line containing the last item name in the entry specification must be terminated with a semicolon. The first item appearing in the entry definition is known as the **search** item, and must have sub-item count of 1 (simple item).

The path count specifies the number of paths to be established to detail data sets. For manual master data sets, the path count must be an integer from 0 thru 8. A manual master set with a path count of 0 is known as a stand alone master set (not associated with a detail set). For automatic master sets, the path count is an integer from 1 thru 8. Automatic master sets cannot stand alone.

The maximum entry count must be an integer from 1 thru 65534. This count specifies the maximum number of entries to be stored in the set. Examples of data set definitions are shown in Chapter 5.

Detail Data Set Definition

Detail data sets are defined using the format:

```

{ NAME: } set name , { DETAIL } (read list/write list) [ , volume name ] ;
{ N:   }

{ ENTRY: } item name [ (master set name) ] ,1
{ E:     }
        [item name [ (master set name) ] ;]
        [item name [ (master set name) ] ;]
        .
        .
        .
        [item name [ (master set name) ] ;]

{ CAPACITY: } maximum entry count ;
{ C:         }

```

The set name refers to the detail set being defined. The name is from 1 thru 15 characters, and consists of uppercase alphabetic characters, the numbers 0 thru 9, and the minus sign (-). The first character must be a letter. All set names must be unique within the schema.

¹ If the entry is defined with only one item name, the ENTRY statement is terminated with a semicolon instead of a comma.

The read list and write list contain lists of user class numbers separated by commas, and are used to determine which user classes have access to the set. Specifying a user class number (an integer from 0 thru 31) in the read list allows that user class to have read access to the set. Specifying a user class number in the write list allows that user to have read and write access to the set. The read list may be null, but the write list must contain at least one user class number.

The optional volume name specifies the disc on which the data set is to reside. This name is from 1 thru 8 ASCII characters. If no volume name is specified, the data set will reside on the same volume as the root file.

The item name is the name of a data item previously defined in the item definition section. Each item name must appear on a new line, and must be unique within the data set. A data entry may be defined with up to 127 item names. The line containing the last item name in the entry specification must be terminated with a semicolon.

The master set name refers to a previously defined master data set. When a master set name follows an item name, it indicates that the data item is a search item linking the detail data set to the named master set. Up to 8 data paths may be defined. If no data paths are defined in the detail set, the set is known as a stand alone detail set.

In order for a data path between a master and a detail set to be valid, the search item type (I,S,L or X) in each set must be the same. For string items (type X), the string lengths must also be the same. The search item name in the master set does not have to match the search item name in the detail. Only simple items (sub-item count equal to 1, or not specified) can be search items.

Media Record Length

IMAGE/250 transfers all records to and from a disc location in the form of a media record. This media record consists of an entire data entry and all related path pointers. The media record length of a data entry (defined in the data set definition section) cannot exceed 1024 bytes. The media record length is equal to the sum of all the items used to define the data entry (entry length) plus the length of all data chain pointers.

For master data sets, the data chain pointer length equals six bytes, plus six bytes for each path defined in the master set. For detail data sets, the data chain pointer length equals four bytes per path, or two bytes for stand alone detail sets. In all cases, the minimum media record length is six bytes. Some examples are shown below.

Set Type	Entry Length (in bytes)	Number of Paths	Media Record Length (in bytes)
Master	200	0	206
Master	200	5	236
Detail	2	0	6
Detail	4	0	6
Detail	6	0	8
Detail	200	0	202
Detail	200	5	220

The total number of bytes of disc space required to store a data set is the product of the media record length and the maximum number of entries. This number should not exceed the maximum space available on the medium used to store the data set. The number of sectors required to store the data set (equal to the product of the media record length and the maximum number of entries divided by 256) cannot exceed 65534 sectors.

Schema Commands

The data base definition (schema) can include statements to select Schema Processor options and to specify page control. Each statement must appear on a separate line, and may appear anywhere within the schema. These statements, known as Schema commands, cannot contain comments.

If a parameter list is included with the command, it must be separated from the command name by at least one blank. Parameters must be separated by commas. Blanks may be freely inserted between items in the parameter list.

The \$TITLE Command

The \$TITLE command specifies a character string to be printed at the top of each new page of the schema listing. It does not cause a page eject.

```
$TITLE [ "character string " ]
```

The title specified by the character string overrides any titles specified by previous \$TITLE or \$PAGE commands. If the character string is omitted, no title will be printed until a subsequent \$TITLE or \$PAGE command specifies one. The quote character (") may be specified in the character string by a pair of quotes (" "). Title strings longer than 30 characters are truncated. Some examples are:

Command	Title Printed
\$TITLE	
\$TITLE "Inventory report"	Inventory Report
\$TITLE "" "Master Data Set" ""	"Master Data Set"

The \$PAGE Command

The \$PAGE command causes the schema listing to eject to the top of a new page, unless the NOLIST option has been selected by a previous \$CONTROL command. The \$PAGE command is not listed.

```
$PAGE [ "character string " ]
```

If a character string is specified, the string replaces the title string specified by a previous \$TITLE or \$PAGE command. If no character string is specified, the title string is unchanged. The quote character (") may be specified in the character string by a pair of quotes (" "). Title strings longer than 30 characters are truncated. Some examples of this command are:

Command	Title Printed
\$PAGE	
\$PAGE "Detail Data Set"	Detail Data Set
\$PAGE "" "Detail Data Set" ""	"Detail Data Set"

	error occurred is listed, followed by an error message.
ROOT	Causes the Schema Processor to build a root file if no errors are detected in the schema.
NOROOT	Prevents the Schema Processor from building a root file.
TABLE	Causes the Schema Processor (if no preceding errors are detected) to print a table containing data set information following the listing. The information printed includes data set name, type, number of fields, number of paths, entry length, media record length, capacity, number of sectors, and volume labels.
NOTABLE	Suppresses the TABLE option, preventing the data set information table from being printed.
ERRORS=nnn	Sets the maximum number of allowed errors equal to nnn. If this number is exceeded during processing, the Schema Processor terminates immediately. The number must be an integer value from 0 thru 999.
LINES=nnn	Sets the number of lines to be printed on a page. The number must be an integer from 20 thru 999.

The \$CONTROL options can be placed in any order, but each must be separated by a comma. At least one option must be specified when using \$CONTROL.

Options not redefined by a \$CONTROL command default to:

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

These default options are equivalent to using the \$CONTROL command:

```
$CONTROL LIST, ROOT, TABLE, ERRORS = 100, LINES = 66
```

Some other examples of the \$CONTROL commands are:

```
$CONTROL NOROOT, TABLE  
$CONTROL ERRORS = 20
```

Schema Processor Operation

The Schema Processor is a program used to create the data base root file. Once the data base has been defined using the DBDL, a data file containing the definition is created using the text EDITOR program (See Appendix B). This data file is used by the Schema Processor to generate a listing of the data base definition and to produce the root file.

The Schema Processor program is located on the SYSTEM disc. The program is divided into three parts, having the program file names SCHEMA, SCHOV2, and SCHOV3. The disc containing these files must be on-line during Schema Processor operation. An error message file, SCHERR, must be on the same disc as the Schema program files. A temporary file, which requires 120 sectors, is created on the same disc, and is later purged during normal operation. The temporary file name is \$SCH\$x, where x is an integer from 1 thru 6.

Using the Schema Processor

The Schema Processor generates a listing of the data base definition, along with error messages and a summary table. This listing will be directed to the selected printer. The default printer is the 8. To change the printer, press the CHANGE PRINTER softkey and enter the select code. The listing can be suppressed by entering a code of 9. The default key transformation (hashing algorithm) is STANDARD. To change the hashing algorithm to PRE-OS6, press the KEY TRANSFORMATION softkey.

To run the Schema Processor program execute the command:

```
RUN "SCHEMA [volume spec]"
```



The volume spec parameter must appear when the SCHEMA program is not on the default mass-storage device. The Schema Processor will operate correctly regardless of which disc drive contains the Schema programs and which drive has been designated to be the default disc drive (using the MSI command).

Following the RUN command, the Schema Processor displays this menu:

```
SCHEMA PROCESSOR
```

```
KEY TRANSFORMATION IS STANDARD                                PRINTER IS 8
```

```
Please enter the name of the schema text file.
```

CHANGE PRINTER	KEY TRANSFORM					EXIT PROGRAM
-------------------	------------------	--	--	--	--	-----------------

The default key transformation (hashing algorithm) is STANDARD. To change the hashing algorithm to PRE-OS6, press the KEY TRANSFORM softkey.

The program requests the name of the data file containing the data base definition (schema text file). If this file has been created by the EDITOR program, it may be in either numbered or unnumbered format. The data file should be entered in the form:

file name [volume spec]
or
"file name [volume spec]"

The data file containing the data base definition must be on-line before the file name is entered. Processing will begin immediately following the entry of the schema text file name. The Schema Processor may be terminated at any time by pressing either **HALT** or the EXIT softkey.

Routine Messages

The Schema Processor prints a heading on the listing which includes the product name IMAGE/250 and the product identification number. Subsequent pages are headed by a page number, the data base name, and the title most recently specified by a \$TITLE or \$PAGE command. All page headings are followed by two blank lines.

If the LIST option has been selected (using the \$CONTROL statement or by default), a copy of the data base definition is printed on the default printer. If the NOLIST option has been selected (using the \$CONTROL statement), only source lines containing errors are printed. In either case, all error messages are of the form:

***** ERROR ***** error message

and follow the schema line containing the error. A blank line is printed between the error message and the next schema line.

After the entire schema has been scanned, several types of summary information can be printed. If no errors have been found, and if there are any data items which were defined in the item part of the schema but not used in the set part, the message:

UNREFERENCED DATA ITEMS :

is printed, followed by the names of the unreferenced items. Their existence is not considered an error, but these items should be removed from the item part to reduce the size of the tables in the root file and the data segment used by IMAGE/250 statements.

If no errors have been found, and if the TABLE option has not been overridden by a \$CONTROL NOTABLE command, a data set information table is sent to the default printer. For each data set, the information contained in the table, from left to right, is as follows:

Data Set Name

Data Set Type –

- Manual Master
- Automatic Master
- Detail

Field Count – The number of items in each entry.

Path Count – The number of data paths associated with the data set.

Entry Length – The total length, in bytes, of the data items in each entry of the data set.

Media Record Length – The total length, in bytes, of each media (physical) record in the set; this includes the data entry plus all related path pointers.

Data Set Capacity – Maximum number of entries allowed in the data set.

Sectors – The number of 256-byte sectors of storage space required for the data set.

Volume Name – Name of the disc on which the data set is to reside after it is created. If no volume name is listed, the data set will reside on the root file volume.

The root file length and the total number of 256-byte sectors required by the data base are also listed with the TABLE.

Two lines of summary information appear at the end of the schema listing. The total number of error messages printed is listed on the first line. The second line contains the total number of data items and data sets defined in the schema. The local and global data base control block (DBCBC) lengths are printed when no errors are detected. An additional line is printed below these lines if the ROOT option has not been overridden by a \$CONTROL NOROOT command. This line will be one of these messages:

```
PRECEDING ERRORS -- NO ROOT FILE CREATED
or
ROOT FILE data base name GENERATED.
```

An example schema listing is shown in Chapter 5.

NOTE

Data bases having a combined global and local DBCBC length greater than 4240 bytes may require more than 32k bytes of user memory for the DBUNLD and DBLOAD utilities to operate. (See chapter 4.)

CHAPTER 3

Data Base Manipulation

Introduction

This chapter introduces the IMAGE/250 manipulation statements. A functional description of each statement is also provided. Since a working knowledge of the BASIC/250 language is assumed throughout this chapter, refer to the HP 250 BASIC Programming Manual when necessary.

The following list summarizes the manipulation statements. For example programs using these statements, refer to Chapter 5. The syntax conventions used in this chapter are the same as those described in Chapter 2.

IMAGE/250 Summary

DBOPEN	Initiates access to a data base. Sets up the access mode and user-class number for the specified data base.
DBCLOSE	Terminates access to a data base.
DBGET	Reads the data items of a specified entry in a data set.
DBUPDATE	Modifies specified item values in an entry. (Search items may not be modified.)
DBPUT	Adds new entries to a data set.
DBDELETE	Deletes existing entries from a data set.
DBFIND	Locates the first and last entries of a data chain in a detail data set in preparation for access to that chain.
DBINFO	Provides structural data base information such as data item names, data set names, and field descriptions.
DBLOCK	Locks data base records to allow the user exclusive write access.
DBUNLOCK	Unlocks data base records locked with a previous DBLOCK.
DBASE IS	Defines the data base to be used prior to the IN DATA SET statement.
IN DATA SET	Automatically packs the buffer parameter during DBPUT and DBUPDATE. Automatically unpacks the buffer after DBGET.
PREDICATE	Defines the data base records to be locked via DBLOCK.

The DBOPEN Statement

DBOPEN initiates access to a data base, setting up the access mode and user-class number.

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

The parameters are:

base name	A string variable containing the data base name preceded by two blank spaces. An optional volume spec may follow the data base name.
password	A string expression containing a left-justified ASCII string.
mode	A numeric expression equal to 1, 3 or 8.
status	An integer array variable that returns status information after DBOPEN is executed. The array must contain at least ten elements in its right-most dimension.

DBOPEN Modes

A data base may be opened in one of three modes. These modes determine the type of operations that can be performed by all users accessing the data base.

Mode 1: modify shared with data base locking. Data entries may be read and written within the constraint of the user-class number granted by DBOPEN. Data bases opened in mode 1 must be locked (DBLOCK) before data set entries may be added, deleted, or modified.

Mode 3: modify exclusive. DBLOCK and DBUNLOCK statements are not required in this mode. Exclusive access is obtained for reading and/or writing. Although the data base may change in content, it does so under your exclusive control. Control is relinquished after executing a DBCLOSE operation. Other requests for access to the data base are refused until a DBCLOSE operation is executed.

Mode 8: read shared. The data base is opened for shared read access. Writing to the data base is not permitted.

If successful, DBOPEN replaces the first two characters of the base name string variable, formerly blanks, with an ASCII data base number between 00 and 04. This is the internal ID number of the data base and should not be altered.

A corrupt data base is accessible in mode 8. Status error -94 is returned if the data base is corrupt.

DBOPEN Status Array

A DBOPEN error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appear in Appendix E. The following table describes the status array contents after a successful DBOPEN.

DBOPEN Status Array

Array Element	Value	Description
1	0	CW.
2	0 thru 31	User class number.
3	Shared DBCB	Word length of shared DBCB.
4	Local DBCB	Word length of local DBCB.
5	0	
6	Bits 0 thru 11	The DBOPEN identification number (401).
	Bits 12 thru 15	The mode value used to open the data base.
7	Program line number	
8	0	
9	Mode number	DBOPEN mode parameter value (same as bits 12 thru 15 of element 6).
10	0 thru 10000	For HP use only.

The DBCLOSE Statement

DBCLOSE terminates access to the specified data base.

```
DBCLOSE (base name , data set , mode , status )
```

The parameters are:

base name	The same string variable used when opening the data base.
data set	Any string or numeric expression.
mode	A numeric expression equal to 1, 3 or 4.
status	An integer array variable that returns status information after DBCLOSE is executed. The array must contain at least ten elements in its right-most dimension.

DBCLOSE Modes

Two modes are available when closing a data base.

Mode 1: close the data base. The data base described is closed, the memory segment assigned by DBOPEN is released, and the first two bytes of the base name parameter are reset to blanks. The data set parameter is ignored.

Mode 3: data set rewind. The specified data-set pointer is reset to the first item in the set.

Mode 4: update the data base with the current control and record information. Data base modifications alter information in the user's Data Base Control Block. The user's private data base buffer may also contain modified data that has not been transferred to the disc file. A mode 4 DBCLOSE updates the root file and the appropriate data set with the current information in the user's memory. The data base status is unchanged and further access is allowable. The data set parameter is ignored.

DBCLOSE Status Array

A DBCLOSE error assigns a non-zero condition word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBCLOSE.

DBCLOSE Status Array

Array Element	Value	Description
1	0	CW.
2 thru 4	Unchanged	
5	0	
6	403	The DBCLOSE identification number.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	0 thru 10000	For HP use only.

The execution of certain system commands cause an implicit DBCLOSE to be performed. Some commands, such as STOP or PAUSE, perform a mode 4 DBCLOSE on all data bases currently open by the user. This operation copies updated information to the disc if required, but leaves the data base open for further access. Other commands, such as RUN or SCRATCH C, perform a mode 1 DBCLOSE on all data bases currently open by the user. This terminates access to the data base until a DBOPEN is executed. The following table describes what statements perform an implicit DBCLOSE.

Condition or Command	DBCLOSE Operation ¹
STOP or END	Mode 4
ERROR	Mode 4
PAUSE or HALT (step)	Mode 4
RUN	Mode 1
SCRATCH C	Mode 1
SCRATCH A or CTRL HALT	Mode 1

¹ If the required volumes are not mounted, the DBCLOSE is not performed. Any mass memory errors encountered during the operation are either displayed or trapped by ON ERROR.

The DBGET Statement

DBGET reads the specified data set entry into a string variable.

```
DBGET (base name , data set , mode , status , list , buffer , argument )
```

The parameters are:

base name	The same string variable used when opening the data base.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 2, 4, 5 or 7.
status	An integer array variable that returns status information after DBGET is executed. The array contains at least ten elements in its right-most dimension.
list	A string expression containing @ or @: or @. This value means that only the entire entry can be accessed and is referred to as the full record mode.
buffer	A simple string variable in which DBGET returns the specified record entry. The maximum buffer length must equal or exceed the data-set entry length.
argument	Direct access (mode 4): A numeric expression representing a record number. Calculated access (mode 7): An expression of the same data type as the master data set's search item.

DBGET Modes

DBGET is used to read entries from the various data sets in a data base. The mode parameter determines the type of access requested: serial, directed, chained, or calculated.

Mode 2: serial read. DBGET serially searches the specified data set for the next non-empty record. The value of the argument parameter is ignored for this mode.

Mode 4: directed read. DBGET examines the record located at the address contained in the argument parameter. If the record is not empty, the entry is copied into the buffer. An error condition is returned in the first word of the status array if the record is empty.

Mode 5: chain read. The next entry of the current chain of the specified master or detail set is read. DBFIND is used to set the current chain pointer for a detail data set. Any DBGET mode may be used to set the current record pointer for a synonym chain within a master set. The value of the argument parameter is ignored for this mode.

Mode 7: calculated read. This mode is used with master data sets only. The entry with a search item value matching the argument parameter is copied into the buffer. If the search item is numeric, the numeric argument will be converted to the proper type (integer, short, or real) before the calculated read is performed.

IMAGE/250 supports the full record mode of data transfer (list equals @: or @). Thus, in all modes, the data items read are those which represent the entire data entry.

DBGET Status Array

A DBGET error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBGET.

DBGET Status Array

Array Element	Value	Description
1	0	CW.
2	Buffer Length	Number of words transferred to the buffer.
3	0	
4	Record Number	Integer number of accessed record.
5	0	
6	Synonym chain length or 0	
7	0	
8	Backward Address	Integer address of the previous record in a chain.
9	0	
10	Forward Address	Integer address of the next record in a chain.

For a detail data set, the forward and backward addresses are always updated relative to the path established by the previous DBFIND applied to the data set.

For a master data set, the values of elements 6, 8, and 10 of the status array depend on whether a primary or secondary entry was accessed. For primary entries, these elements represent the synonym chain count, the address of the last member of the synonym chain, and the address of the first member of the synonym chain. For secondary entries, the sixth array element is 0, while the eighth and tenth elements represent addresses to the previous and next members of the synonym chain.

Record numbers, chain lengths, and forward and backward record addresses fall into the range of -32768 or 32767 . These integers correspond to actual positive pointers or lengths as shown in the following table:

Integer	Positive
0	0
1	1
.	.
.	.
.	.
32767	32767
-32768	32768
.	.
.	.
.	.
-2	65534

This statement may be useful in converting integers:

$$\text{Positive} = \text{Integer} + 65536 * (\text{Integer} < 0).$$

The DBUPDATE Statement

DBUPDATE modifies item values in a data entry.

```
DBUPDATE (base name , data set , mode , status , list , buffer )
```

The parameters are:

base name	The same string variable used when opening the data base.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBUPDATE is executed. The array must contain at least ten elements in its right-most dimension.
list	A string variable containing @ or @; or @. This value means that only entire records can be accessed and is referred to as the full record mode.
buffer	A simple string variable containing data that is to replace the current data entry in the specified set.

Since search items cannot be updated, the search item value must match the present buffer value for that item. Before updating a data entry, DBUPDATE must be preceded by a DBGET or DBPUT to establish the current record pointer.

The data item values of the current data entry are replaced using the data supplied. Since data is only transferred in the full record mode, the buffer must contain all the values for the items contained in the data set entry. The search item values in the buffer must match the old values of the current record or the update is not performed.

DBUPDATE Status Array

A DBUPDATE error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. For a successful DBUPDATE, the status array contents are as described under DBGET, except that the second element represents the number of words transferred from the buffer to the data set.

The DBPUT Statement

DBPUT adds new data entries to a manual master or detail set.

```
DBPUT (base name , data set , mode , status , list , buffer )
```

The parameters are:

base name	The same string variable used when opening the data base.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBPUT is executed. The array must contain at least ten elements in its right-most dimension.
list	A string variable containing @ or @; or @. This value means that only entire records can be accessed and is referred to as the full record mode.
buffer	A simple string variable containing data that is to be placed into the specified set.

The buffer parameter must contain values for all items in the data entry to be added. IMAGE/250 determines the physical record placement of the entry within the data set.

When the data set is a manual-master data set, IMAGE/250 verifies that:

- The master data set is on-line.
- The master data set is not full.
- No existing data entry has a search item value identical to the new data entry.

If any of these tests fail, an error condition is returned in the status array. Otherwise, the new data entry is added to the primary address, if possible, and at a secondary address, if not. In the latter case, it is attached to the end of a synonym chain.

When the data set parameter is a detail data set, IMAGE/250 verifies that:

- The detail data set and all related master data sets are on-line.
- The detail data set is not full.
- The related manual-master data sets have entries with search item values that match the corresponding detail search item values.
- All necessary entries in the related automatic masters are present or, if not, that the related sets are not full.

If any of these tests fail, an error condition is returned in the first element of the status array. Otherwise, the data entry is added to the detail data set with IMAGE/250 performing all linkage maintenance and, when necessary, creating entries in the automatic-master data sets.

IMAGE/250 determines the physical location of the entry within the data set. If the detail data set is related to one or more master data sets, however, the entry is logically linked to the end of each chain.

Data may not be added directly (DBPUT) to automatic masters. Data is added or deleted automatically as detail data set entries are added or deleted.

DBPUT Status Array

A DBPUT error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. For a successful DBPUT, the status array contents are as described under DBGET, except that the second element represents the number of words transferred from the buffer to the data set.

The DBDELETE Statement

DBDELETE deletes existing data entries in a manual-master or a detail data set.

```
DBDELETE (base name , data set , mode , status )
```

The parameters are:

base name	The same string variable used when opening the data base.
data set	Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBDELETE is executed. The array must contain at least ten elements in its right-most dimension.

When deleting entries from master data sets, all pointer information for chains indexed by the entry must indicate that the chains are empty. In other words, there must not be any detail entries on the paths defined by the master which have the same search item value as the master entry to be deleted.

When deleting detail set entries, IMAGE performs the required changes to chain linkages and other chain information, including the chain heads in related master data sets. If the last member of each detail chain linked to the same automatic master has been deleted, DBDELETE also deletes the master entry containing the chain heads.

DBDELETE Status Array

A DBDELETE error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBDELETE.

DBDELETE Status Array

Array Element	Value	Description
1	0	CW.
2	Buffer Length	Number of words transferred in the buffer.
3	0	
4	Record Number	Address of deleted record.
5	0	
6	Chain Length	
7	0	
8	Backward Address	See below.
9	0	
10	Forward Address	See below.

For detail data sets, the chain length value is 0. For master data sets, this number is 0, unless the deleted entry was a primary entry with synonyms. In this case, the number is one less than its previous value.

Words 8 and 10 are the unchanged backward and forward addresses of a chain. For master data sets, if the new synonym chain count is non-zero, the numbers reference the last and first synonym chain entries respectively.

The DBFIND Statement

DBFIND is used only with detail data sets, and locates the head of a chain in a master data set whose search-item value is identified by the argument parameter.

```
DBFIND (base name, data set, mode, status, item, argument)
```

The parameters are:

base name	The string variable used when opening the data base.
data set	Either a string variable containing a left-justified data set name or a numeric variable containing a data set number corresponding to the data set's position in the schema definition.
mode	A numeric expression equal to 1.
status	An integer array variable that returns status information after DBFIND is executed. The array must contain at least ten elements in its right-most dimension.
item	A string expression containing a left-justified search item name, or a numeric expression containing a search item number. The search item number represents the relative position in the (item part) schema definition. This number is an integer ranging from 1 thru 255. The specified search item defines the path to which the chain belongs.
argument	Contains a value for the search item to be used during calculated access. DBFIND uses this value to locate the desired chain head in the master data set. The argument and search item data types must match. DBFIND converts numeric arguments to the search item numeric type during execution.

DBFIND verifies that the item parameter references a search item for the specified detail data set. It then locates the appropriate master data set entry whose search item value (or key) matches the value of the argument parameter. The internal status information relative to the data set parameter is adjusted in anticipation of subsequent chained references to that same data set (DBGET, mode 5). Note that DBFIND does not retrieve data entries; it simply establishes the current record pointer.

DBFIND Status Array

A DBFIND error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBFIND.

DBFIND Status Array

Array Element	Value	Description
1	0	CW.
2	0	
3	0	
4	0	
5	0	
6	Chain Length	Integer count of entries in the current chain.
7	0	
8	End of Chain Address	Integer address of the last record in the chain.
9	0	
10	Chain Head Address	Integer address of the first record in the chain.

The DBINFO Statement

DBINFO provides data base structural information from the root file and does not access information within the data sets.

```
DBINFO (base name , qualifier , mode , status , buffer )
```

The parameters are:

base name	The same string variable used when opening the data base.
qualifier	A variable that references a data set, data item, or volume either by name or number (see the next tables).
mode	A numeric expression specifying the type of information to be returned.
status	An integer array variable that returns status information after DBINFO is executed. The array must contain at least ten elements in its right-most dimension.
buffer	A string variable long enough for the requested information to be returned. The contents of the buffer varies according to the mode parameter used.

The various types of information requests are divided into four categories: data sets, data items, data paths (i.e., relationships between data sets), and data set volumes. In all cases, the information supplied is dependent on the access mode used, and is restricted by the user-class number established when the data base was opened. Any data sets or data paths in the data base which are inaccessible to that specific user class are considered to be nonexistent by DBINFO.

For each mode, the information is returned to the buffer as a word string. Some of the string information may actually represent numeric data. The UNPACK statement, described in Appendix A, provides a means of properly interpreting these values. The following tables describe the results associated with a successful execution of DBINFO (conditional word equal to 0).

DBINFO Modes Returning Data Item Information

Mode	Purpose	qualifier	buffer	Contents	Comments						
101	Identifies the data item number for a given data item.	data item name or number	word 1	data item number	Integer.						
102	Describes a specific data item.	data item name or number	word 1 . . 8 9 10 11 12 13	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">data item name</td></tr> <tr><td style="padding: 2px;">data type Δ</td></tr> <tr><td style="padding: 2px;">item-word length</td></tr> <tr><td style="padding: 2px;">sub-item count</td></tr> <tr><td style="padding: 2px;">0</td></tr> <tr><td style="padding: 2px;">control number</td></tr> </table>	data item name	data type Δ	item-word length	sub-item count	0	control number	Left-justified and filled with blanks. (L,S,I,X) Δ indicates blank. Integers.
data item name											
data type Δ											
item-word length											
sub-item count											
0											
control number											
104	Identifies all data items in a specific data set. The data items are listed in the order of their occurrence in the data entry.	data set name or number	word 1 2 . . . n+1	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px; text-align: center;">n</td></tr> <tr><td style="padding: 2px;">data item number</td></tr> <tr><td style="padding: 2px;"> </td></tr> <tr><td style="padding: 2px;"> </td></tr> <tr><td style="padding: 2px;"> </td></tr> <tr><td style="padding: 2px;">data item number</td></tr> </table>	n	data item number				data item number	n=number of data items listed. All words are integers.
n											
data item number											
data item number											

DBINFO Modes Returning Data Set Information

Mode	Purpose	qualifier	buffer	Contents	Comments										
201	Identifies a data set number for a given data set.	data set name or number	word 1	±data set number	If positive, entries can only be read. If negative, entries can be read, written, or modified.										
202	Describes a specific data set.	data set name or number	word 1 : 8 9 10 11 12 13 14 15 16 17	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px;">data set name</td></tr> <tr><td style="padding: 2px;">set type Δ</td></tr> <tr><td style="padding: 2px;">entry word-length</td></tr> <tr><td style="padding: 2px; text-align: center;">0</td></tr> <tr><td style="padding: 2px; text-align: center;">0</td></tr> <tr><td style="padding: 2px; text-align: center;">0</td></tr> <tr><td style="padding: 2px; text-align: center;">0</td></tr> <tr><td style="padding: 2px;">number of data entries</td></tr> <tr><td style="padding: 2px; text-align: center;">0</td></tr> <tr><td style="padding: 2px;">data set capacity</td></tr> </table>	data set name	set type Δ	entry word-length	0	0	0	0	number of data entries	0	data set capacity	Left-justified and filled with blanks. (M,A,D) Δ indicates blank Integers.
data set name															
set type Δ															
entry word-length															
0															
0															
0															
0															
number of data entries															
0															
data set capacity															
203	Identifies all accessible data sets in the data base. The sets are listed in the order of their occurrence in the schema.	(ignored)	word 1 2 : : : n+1	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px; text-align: center;">n</td></tr> <tr><td style="padding: 2px;">±data set number</td></tr> <tr><td style="padding: 2px; text-align: center;">.</td></tr> <tr><td style="padding: 2px; text-align: center;">.</td></tr> <tr><td style="padding: 2px; text-align: center;">.</td></tr> <tr><td style="padding: 2px;">±data set number</td></tr> </table>	n	±data set number	.	.	.	±data set number	n=number of accessible data sets in data base. Arranged in data set number order. If positive, the data set can only be read. If negative, entries can be read, written, or modified. All words are integers.				
n															
±data set number															
.															
.															
.															
±data set number															
204	Identifies all accessible data sets which contain a specified data item.	data item name or number	(Same as mode 203)	(Same as mode 203)	n=number of accessible data sets. Arranged in data set number order. If positive, the data set can only be read. All words are integers.										

DBINFO Modes Returning Data Path Information

Mode	Purpose	qualifier	buffer	Contents	Comments													
301	Identifies paths defined for a specified data set.	data set name or number	word 1 2 3 4 3n-1 3n 3n+1	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="text-align: center;">n</td></tr> <tr><td style="text-align: center;">data set number</td></tr> <tr><td style="text-align: center;">search item number</td></tr> <tr><td style="text-align: center;">0</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">data set number</td></tr> <tr><td style="text-align: center;">search item number</td></tr> <tr><td style="text-align: center;">0</td></tr> </table>	n	data set number	search item number	0	data set number	search item number	0	<p>n=number of paths</p> <p>Repeat for each path. If qualifier refers to master, set number is for detail. If qualifier refers to detail, set number is for master. Item numbers identify items in detail set.</p> <p>Path designators presented in order of their appearance in schema.</p> <p>All words are integers.</p>
n																		
data set number																		
search item number																		
0																		
.																		
.																		
.																		
.																		
.																		
.																		
data set number																		
search item number																		
0																		
302	Identifies a search item for specified data set.	master data set name or number OR detail data set name	word 1 2 word 1 2	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="text-align: center;">search item number</td></tr> <tr><td style="text-align: center;">0</td></tr> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="text-align: center;">search item number</td></tr> <tr><td style="text-align: center;">data set number</td></tr> </table>	search item number	0	search item number	data set number	<p>Search item number in master set. All words are integers.</p> <p>First search item defined in the detail data set.</p> <p>All words are integers.</p>									
search item number																		
0																		
search item number																		
data set number																		



DBINFO Modes Returning Volume Information

Mode	Purpose	qualifier	buffer	Contents	Comments						
401	Identifies a volume number for a given data set.	data set name or number	word 1	± volume number	If the volume number is positive, the volume is not mounted. If negative, the volume is mounted. All volume numbers are integers A 0 volume number denotes the data set is mounted on the root file volume.						
402	Identifies a volume name for a given volume.	volume name or number	words 1 thru 4	volume name	Left-justified and filled with blanks, if necessary.						
403	Identifies all volumes for a given data base.	(ignored)	word 1 2 n+1	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="text-align: center;">n</td></tr> <tr><td style="text-align: center;">± volume number</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">± volume number</td></tr> </table>	n	± volume number	.	.	.	± volume number	n=number of volumes for the data base If the volume number is positive, the volume is not mounted. If negative, the volume is mounted. All words are integers When n=0, the entire data base is located on the root file volume.
n											
± volume number											
.											
.											
.											
± volume number											
404	Identifies all the data sets on a given volume.	volume name or number	word 1 2 n+1	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="text-align: center;">n</td></tr> <tr><td style="text-align: center;">± data set number</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">.</td></tr> <tr><td style="text-align: center;">± data set number</td></tr> </table>	n	± data set number	.	.	.	± data set number	n=number of data sets. If the data set number is positive, data entries can only be read. If negative, data entries can be read, written, or modified. All words are integers
n											
± data set number											
.											
.											
.											
± data set number											

DBINFO Status Array

A DBINFO error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBINFO.

DBINFO Status Array

Array Element	Value	Description
1	0	CW.
2	Buffer Length	Number of words transferred to the buffer.
3	0	
4	Unchanged	
5	0	
6	Bits 0 thru 11	The DBINFO identification number 402.
	Bits 12 thru 15	The mode value used to open the data base.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	0 thru 10000	For HP use only.

The DBLOCK Statement

DBLOCK locks all or part of a data base and provides either exclusive write access or read access which excludes all write access.

```
DBLOCK (base name , qualifier , mode , status )
```

The parameters are:

base name	The same string variable used when opening the data base.
qualifier	A variable which references a data set or a string expression that describes the lock to be applied (see table below).
mode	A numeric expression defining type of lock (see table below).
status	An integer array variable that returns status information after DBLOCK is executed. the array must contain at least ten elements in its rightmost dimension.

MODES	QUALIFIER	COMMENTS
1,2,11,12	Ignored	Data Base level locks.
3,4,13,14	Data set name or number	Data Set level locks.
5,6,15,16	String expression lock descriptor	General-purpose lock to entry level.

DBLOCK locks the section requested if no other user currently has a conflicting lock. If the lock cannot be granted immediately, the request is placed in a queue to wait for access. Access is granted only after all conflicting locks ahead in queue have been granted and released: (Although the section requested is not locked but is in queue, no other request which would lock a subsection and hold up the first request is granted. Refer to Data Base Locking in chapter 5.)

Even mode numbers request a lock without wait; if the lock cannot be granted immediately, no lock is made. The reason for the lock failure is indicated in the condition word of the status array. The request is not queued.

If a program has a lock in effect, additional locks can be made only with even modes. Using the "no-wait" modes prevents successive lock requests from causing a deadlock error. A status error is returned if an odd mode is used with a lock already in effect.

A write lock is exclusive. No other locks may be made on that section. A read lock may be made on a section that already has a read lock.

Summary of DBLOCK Modes

Mode	Wait	Type of Access	
		Write	Read
1	Yes	Entire data base	Entire data base
3		Data set	
5		Predicate	
11			
13			
15			
2	No	Entire data base	Entire data base
4		Data set	
6		Predicate	
12			
14			
16			

For example, this statement:

```
100 DBLOCK (Base$, "PARTS", 4, Status(*))
```

request without wait a write lock on the data set PARTS. This statement:

```
110 DBLOCK (Base$, P$&Q$, 15, Status(*))
```

requests with wait a read lock on the data sets and data entries specified in the string formed by the concatenation of the lock descriptors in P\$ and Q\$.

DBLOCK Status Array

A DBLOCK error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBLOCK.

DBLOCK Status Array

Array Element	Value	Description
1	Conditional word	0 for successful lock.
2	Descriptor number	Number causing failure.
	1	For successful lock in modes 1 thru 4.
3	0	If CW=20, data base locked.
	1	If CW=20, data set or data entries locked.
4	reserved	
5	0	
6	Bits 0 thru 11	The DBLOCK identification number 409.
	Bits 12 thru 15	The mode value used to open the data base.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	0 thru 10000	For HP use only.

The DBUNLOCK Statement

DBUNLOCK unlocks the data base and returns it to a multi-access state.

```
DBUNLOCK (base name, qualifier, mode, status)
```

The parameters are:

- base name The same string variable identifying the data base name when the data base was opened.
- qualifier Any string or numeric expression. Unused by DBUNLOCK.
- mode A numeric expression equal to 1.
- status An integer array variable that returns status information after DBUNLOCK is executed. The array must contain at least ten elements in its right-most dimension.

DBUNLOCK relinquishes locks to all sections of the data base previously acquired by DBLOCK. If DBUNLOCK is executed at a time when the user does not have a lock, no error is returned.

DBUNLOCK Status Array

A DBUNLOCK error assigns a non-zero conditional word (CW) to the first element of the status array. A list of all CW values and their meaning appears in Appendix E. The following table describes the status array contents after a successful DBUNLOCK.

DBUNLOCK Status Array

Array Element	Value	Description
1	Conditional word	0 for successful unlock.
2 thru 4	Unchanged	
5	0	
6	Bits 0 thru 11	The DBUNLOCK identification number 410.
7	Program line number	
8	0	
9	Mode number	The mode parameter value.
10	0 thru 10000	For HP use only.

Advanced Access Statements

Two advanced access statements are described on the following pages. These statements facilitate the use of the **IMAGE/250** programming statements discussed earlier by loading or unloading information during **DBGET**, **DBPUT**, or **DBUPDATE** execution.

The **DBASE IS** Statement

The **DBASE IS** statement defines the data base to be referenced by subsequent **IN DATA SET** and **WORKFILE IS**¹ statements.

```
DBASE IS base name
```

The parameter is:

base name The same string variable identifying the data base name when the data base was opened.

DBOPEN must be executed prior to executing the **DBASE IS** statement. The data base being referenced is reset either by specifying another **DBASE IS** statement or by closing the referenced data base in mode 1.

¹ For a description of the **WORKFILE IS** statement, refer to the **SORT/250 Programming Manual**.

The IN DATA SET Statement

Through IN DATA SET, data may be automatically transferred from program variables to the buffer variable prior to DBPUT and DBUPDATE. Data may also be automatically transferred from the buffer to program variables after DBGET.

```
IN DATA SET data set [ IN COM] { USE ALL  
                                { USE item list }  
                                { DIM ALL }
```

```
IN DATA SET data set [ IN COM] USE REMOTE LISTS line id list  
                                with  
                                IN DATA SET LIST item list
```

```
IN DATA SET data set FREE
```

The parameters are:

- data set Either a string expression containing a left-justified data set name or a numeric expression containing a data set number corresponding to the data set position in the schema definition.
- item list A group of numeric, string, or array variables used to associate data items to BASIC variables. If there are more items in the schema entry than variables in the list, the extra items are skipped. This can also be done by specifying `SKP` in the item list. The variable names must be in the order as defined by the schema definition, and must be separated by commas.
- line id list A list of line numbers or line labels separated by commas.

During IN DATA SET execution, the data set must be a member of the data base most recently referenced through DBASE IS. Once the item list has been established for a data set, the default data base may be changed through DBASE IS. The correct data-set/data-base relationship is maintained during DBGET, DBPUT, and DBUPDATE operations.

In the USE ALL mode, all program variables are searched for a match against the data set's item names, as stored in the root file. Before a match can be determined the schema names undergo the following conversion:

- All letters except the first are converted to lowercase.
- All dashes (-) are converted to underscores (_).
- If the schema item is a string, a dollar sign (\$) is appended to the name.

The converted schema name is then compared against all program variables. In addition to a name match, a match must exist for:¹

- The data item types.
- The dimension types (simple or one-dimensional array).
- In the case of strings, the maximum string length.

If all of these conditions are satisfied, the program variable is automatically updated when a data set entry is read into the buffer by DBGET. Before an entry is added (DBPUT) or modified (DBUPDATE), data is automatically copied from the program variable to the buffer.

DIM ALL, like USE ALL, establishes a relationship between program variables and matching data set item names. If a match is not found, however, the data set's field is not treated as a skip field, as with the USE ALL mode. Instead, a variable is automatically created within the program with attributes (field type and length) and name matching the corresponding data item.

If an item list is specified, the variables in the list must exactly match the type and length of the corresponding fields in the referenced data set. Comparisons are made on a positional basis; that is, the first data set field is compared to the first variable in the list, the second field to the second variable and so on.

The SKP option may be used to skip unwanted fields. To skip two or more fields in a row, an appended integer may be used (SKP2). This option is useful where only selected fields of a data set entry need to be accessed. Skipped fields are not modified by DBUPDATE, and are assigned a null value by DBPUT. If a particular value is desired in a skipped field, this value can be assigned to the string before the DBPUT or DBUPDATE. Note that assigning a value to the buffer string does not change the value of the program variables specified in the IN DATA SET statement.

USE REMOTE LISTS is a means of referencing item lists that appear in other program lines. The lines referenced in the line id list (either by number or label) contain the actual item list. The item lists are evaluated in the order of the line id list. This option is a method of extending an item list length beyond the maximum program line length of 160 characters.

¹ If the variable was not previously defined during program execution, it will be dimensioned according to the schema item definition.

If the IN COM option is selected, an IN DATA SET statement executed in the main program remains active across all subprograms and functions. IN COM may be used with any of the previously described IN DATA SET options. However, all variables referenced must be explicitly dimensioned in common. This option can only appear in the main program, and not in subprograms or functions.

The FREE option releases the internal relationship between the data set and the program variables. Data is no longer transferred to and from the program variables. If a new IN DATA SET relationship is desired, the FREE option must first be executed. Program execution continues without an error if the FREE option is executed and an IN DATA SET relationship was not established on the data set. If an IN DATA SET relationship exists and the IN COM option was not selected, the FREE option is implied when the current program environment is exited.

NOTE

Care must be taken when executing DBGET to establish the current record pointer for DBUPDATE. Since IN DATA SET automatically transfers the contents of the buffer to program variables during DBGET, these variables must be updated following the DBGET operation.

The PREDICATE Statement

PREDICATE is provided in the IMAGE2 DROM as an aid in setting up predicate strings. It sets up the qualifier parameter that defines the data base entries to be locked via DBLOCK.

```
PREDICATE predicate FROM set1[ , item1[ , rel op , value]] [ ; set2... [ ; setn... ]]
```

The parameters are:

predicate	A string variable returned by PREDICATE and used as the qualifier parameter in DBLOCK.
set ₁	A string expression specifying the data set to be locked or unlocked.
item ₁	A string expression specifying the data item within set ₁ to be locked.
rel op	A string expression containing a relational operator: = or EQ >= or GE <= or LE
value	A string or numeric expression giving the value of the item to be locked.
set ₂ ...	A second set of expressions defining the next lock descriptor.
set _n ...	The n'th set of expressions defining the next lock descriptor.

PREDICATE does no type-checking on the value parameter. The programmer should be careful to match string values with data base items of string type and numeric values with data base items of numeric type. Any discrepancies will result in a status error from DBLOCK. Blanks within item names and set names are ignored. Where multiple descriptors are specified, the descriptor blocks appear in reverse order in the predicate string.

Each of these examples defines a predicate to request a lock of all data entries in data set TRANS having a data item value of 100:

```
100 PREDICATE P$ FROM "TRANS", "PART-NO", "=", "100"
```

```
100 Set$="TRANS"  
110 Item$="PART-NO"  
120 Relop$="="  
130 Value=100  
140 PREDICATE P$ FROM Set$, Item$, Relop$, Value
```

This statement requests a lock on data sets TRANS and INVENTORY (the @ specifies the entire data set):

```
100 PREDICATE Q$ FROM "TRANS", "@", "INVENTORY", "@"
```

This statement requests a lock on all values of PART-NO greater than or equal to 100 in every data set where PART-NO occurs:

```
100 PREDICATE Q$ FROM "@", "PART-NO", ">=", 100
```


100
100
100

CHAPTER 4

Data Base Utilities



Introduction

The IMAGE/250 utilities create, initialize and purge data base files, and perform various maintenance operations. The utilities consist of statements, binary statements, and run-only BASIC language programs. The data base utilities are:

DBCREATE	This statement creates and initializes the data set files of a data base.
DBERASE	This statement erases data set entries from all or selected data sets.
DBPURGE	This statement purges either specific data set files or the entire data base, including the root file and all data sets.
DBSTORE	This binary statement copies either the data base or selected data sets to a backup file. This backup file may span multiple volumes.
DBRESTORE	This binary statement restores the data base following a system failure using the backup file created by DBSTORE.
DBUNLD	This utility program copies data entries from all or selected data sets to a backup file. Data base structural information is not saved.
DBLOAD	This utility program copies data entries into the data base using the backup file created by DBUNLD.
DBMODS	This utility program allows changing various data base structural information without unloading and reloading stored data.
DBPASS	This binary statement changes the password for a stated user-class number.
DBMAINT	This binary statement changes the maintenance password for a stated data base.
READ DBPASSWORD WRITE DBPASSWORD	These binary statements read and write all user passwords from a specified data base.
XCOPY	This binary statement copies root and data set files from one volume to another.

Data Base Utility Statements

The three IMAGE/250 utility statements, DBCREATE, DBERASE, and DBPURGE, are used to create, erase, and purge selected data set files or entire data bases. Each statement requires exclusive access to the data base (i.e., the data base cannot be open). The root file must be on-line during execution.

NOTE

Beginning with Operating System revision 2.D, the DBCREATE, DBERASE and DBPURGE statements are loaded with either the IMAGEU DROM or the DBUTIL binary program. They were removed from the IMAGE DROM to provide more DROM memory space for applications not needing the utility statements.

The DBCREATE Statement

The DBCREATE statement creates and initializes the data set files of a data base.

```
DBCREATE root file spec [ : maintenance word ] [ : set list  
                        [ : volume spec ] [ : return status ]
```

The parameters are:

root file spec	A string expression identifying the data base name. An optional volume label or unit specifier can be appended to the data base name.
maintenance word	A string expression identifying a security password. This expression can be from 1 thru 16 characters in length.
set list	A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.
volume spec	A string expression identifying a volume name or mass-storage device (unit specifier) to contain the created data sets. Only those data sets that reside on the specified volume are created.
return status	A numeric variable in which an error number is returned (refer to Appendix E). 0 is returned if no error occurs.

The DBCREATE statement creates and initializes the data set files of a data base. DBCREATE is used after the Schema Processor has created the root file. Data sets are created either on the root file volume or on the volume specified in the data base definition (schema). When executed from the keyboard, DBCREATE displays the set number of the set being created.

When a set list is supplied, DBCREATE creates only the sets specified. Sets may be identified by name or number (e.g., "1,2,CUSTOMER"). Alternately, all sets on a particular volume or mass-storage device may be created by specifying a volume spec. If neither a set list nor a volume specifier is supplied, DBCREATE attempts to create all data sets of the data base. In this case, the sets are created in an order determined by the set's volume name and set number.

When executed from the keyboard without a return status parameter, certain errors may be reported by DBCREATE without terminating execution (see Appendix E for a description of these non-fatal errors). However, when DBCREATE is executed from a program, or when it is executed from the keyboard and a return variable is used, the first error encountered terminates execution of the statement. When the return status variable is used, the return variable contains the error number (or 0 if no errors are encountered), but no error message is displayed.

The maintenance word on a data base is defined by the first execution of DBCREATE on the data base. Once defined, the maintenance word prevents unauthorized utility operations on the data base. If no maintenance word is supplied by the initial DBCREATE on a data base, future utility operations are performed only when no maintenance word is specified.

The DBERASE Statement

DBERASE erases all entries of data sets by re-initializing data-set files.

```
DBERASE root file spec[ ; maintenance word] [ ; set list  
; volume spec ] [ ; return status]
```

The parameters are:

root file spec	A string expression identifying the data base name. An optional volume label or unit specifier can be appended to the data base name.
maintenance word	A string expression identifying a security password. This expression can be from 1 thru 16 characters in length.
set list	A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.
volume spec	A string expression identifying a volume name or mass-storage device (unit specifier). Only those data sets that reside on the specified volume are erased.
return status	A numeric variable in which an error number is returned (refer to Appendix E). 0 is returned if no error occurs.

The DBERASE statement erases all entries in data set files. All associated path information in related data sets is also erased. This statement is often used prior to loading data entries with the DBLOAD program. When executed from the keyboard, DBERASE displays either the set number of the set being erased, or the related set number followed by a P when path information is being erased.

When a set list is supplied, DBERASE erases only the sets specified. Sets may be identified by name or number (e.g., "1,2,CUSTOMER"). Alternately, all sets of a data base residing on a particular volume or mass-storage device may be erased by specifying a volume spec. If neither a set list nor a volume specifier is supplied, DBERASE attempts to erase all data sets of the data base. In this case, the sets are erased in an order determined by the set's volume name and set number.

When executed from the keyboard without a return-status parameter, certain errors may be reported by DBERASE without terminating execution (see Appendix E for a description of these non-fatal errors). However, when DBERASE is executed from a program, or when it is executed from the keyboard and a return variable is used, the first error encountered terminates execution of the statement. When the return status variable is used, the return variable contains the error number (or 0 if no errors are encountered), but no error message is displayed.

NOTE

After executing a DBERASE on a detail data set, some entries in related automatic-master sets may have a 0 chain count. These unused entries may cause an automatic master to become full when only a few entries are deleted. Unused entries are deleted when an entry containing the unused search item is placed into a related detail and later deleted, or when the automatic-master is erased using DBERASE.

NOTE

Executing a DBERASE on a master data set erases all chain information linking the master set entries with related detail entries. This erased chain information may cause unexpected errors on subsequent accesses of the data base.

The DBPURGE Statement

DBPURGE purges the data base files of a data base.

```
DBPURGE root file spec [ maintenance word ] [ set list ] [ return status ]
                                     [ volume spec ]
```

The parameters are:

root file spec	A string expression identifying the data base name. An optional volume label or unit specifier can be appended to the data base name.
maintenance word	A string expression identifying a security password. This expression can be from 1 thru 16 characters in length.
set list	A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.
volume spec	A string expression identifying a volume name or mass-storage device (unit specifier). Only those data sets that reside on the specified volume are erased.
return status	A numeric variable in which an error number is returned (refer to Appendix E). A 0 is returned if no error occurs.

The DBPURGE statement purges data set files and the root file of a data base. This statement is used prior to executing a DBRESTORE command, or when restructuring the data base. When executed from the keyboard, DBPURGE displays either the number of the set being purged or an # when the root file is being purged.

When a set list is supplied, DBPURGE purges the data set files of the specified sets. Sets may be identified by either name or number (e.g., "1;2;CUSTOMER"). Alternately, all data set files of a data base residing on a particular volume or mass-storage device may be purged by specifying a volume spec. If neither a set list nor a volume specifier is supplied, DBPURGE attempts to purge all data set files of the data base, and if successful, attempts to purge the root file. In this case, data sets are purged in an order determined by the set's volume name and set number.

When executed from the keyboard without a return-status parameter, certain errors may be reported by DBPURGE without terminating execution (see Appendix E for a description of these non-fatal errors). However, when DBPURGE is executed from a program or when it is executed from the keyboard and a return variable is used, the first error encountered terminates execution of the statement. When the return status variable is used, the return variable contains the error number (or 0 if no errors are encountered), but no error message is displayed.

If data base structural information is lost due to a hardware failure, terminated DBRESTORE, or other error, data set files may become disassociated from the data base. DBPURGE provides the capability to purge data base files of a damaged data base. Specifying an * as the first entry in the set list causes DBPURGE to purge all data set files and root files with the name specified by the root file spec. Only files on either the default mass-storage device or the device specified by the root file spec are purged.

Data Base Binary Utility Statements

The two data base binary statements, DBSTORE and DBRESTORE, are used for data base backup and recovery. These binary statements utilize a backup file (which may span several volumes) to store the contents of the data base or selected data sets. Both statements prompt for inserting data set volumes and backup volumes as needed.¹ The binary statements require exclusive access to the data base (i.e., the data base cannot be open).

Before these statements may be used, the binary file DBSTOR must be loaded into user memory. DBSTOR is stored on the SYSTEM disc, and can be loaded by executing:

```
LOAD BIN "DBSTOR [volume spec] "
```

Once the binary has been loaded, these utility statements may be either executed from the keyboard or entered and executed in a program. Programs using DBSTORE and DBRESTORE may be stored (using the STORE command), and later LOADED or RUN without executing the LOAD BIN statement.

¹ Since these utilities prompt for inserting volumes by volume names, the disc containing the root file should have a non-null volume label.

The DBSTORE Statement

The DBSTORE binary statement copies portions of a data base to a backup file. The backup file may be used to restore the data base following a hardware failure or other error.

```
DBSTORE root file spec [ ; maintenance word ] [ ; set list ] TO file spec [ ON volume list ]
```

The parameters are:

root file spec	A string expression identifying the data base name. An optional volume label or unit specifier may be appended to the data base name.
maintenance word	A string expression identifying a security password. This expression can be from 1 thru 16 characters in length.
set list	A string expression identifying particular data sets. Data sets are specified by either name or number. Set identifiers are separated by commas.
file spec	A string expression specifying the name of the backup file to be created by DBSTORE. IF the ON parameter is not specified, an optional volume label or unit specifier may be appended to the backup file name.
volume list	A string expression used to identify the volume name(s) where the data base is to be copied. Each volume name is separated by a comma.

The DBSTORE binary statement copies the entire data base or selected data sets to a backup file. This statement is used whenever a backup copy of the data base is required. Once a data base has been copied to a backup file, it may be restored to the state at which the DBSTORE was executed by using DBPURGE and DBRESTORE. The backup file may span multiple volumes. When DBSTORE is executed from the keyboard, it displays either the set number of the set being stored or an * when the root file is being stored.

When a set list is supplied, DBSTORE copies only the sets specified to the backup file. Sets may be identified by name or number. If the first entry of the set list is an * (e.g., "*; 1; 2; CUSTOMER"), the root file is also copied to the backup file. If a set list is not specified, the entire data base, including the root file, is copied to the backup file. When no set list is given, the sets are copied in an order determined by the set's volume name and the set number.

When no volume list is supplied, DBSTORE creates the backup file on the default mass-storage device, or on the device appearing in the backup file specifier. If a volume list is supplied, DBSTORE creates the backup file on the first volume specified in the list, ignoring any volume in the backup file specifier. Once the backup file has been created, DBSTORE begins copying the specified sets to the backup file. Requests are automatically made to insert backup volumes and data set volumes as needed.

If there is insufficient space for the entire backup file on the first volume, the file is continued on additional volumes. Additional volume names are obtained from the volume list, if specified. If an insufficient number of volume names are given in the volume list, or if no volume list is specified, DBSTORE requests additional volume names as needed.

The DBRESTORE Statement

The DBRESTORE binary statement uses the backup file to restore a data base to its state at the time that DBSTORE was executed.

DBRESTORE file spec [ON volume spec]

The parameters are:

- file spec A string expression identifying the name of the backup file. An optional volume label or mass-storage unit specifier may be appended to the backup file name.
- volume spec A string expression identifying the volume (label) or mass-storage device (unit specifier) where the root file and sets with a default label are to be stored.

The DBRESTORE binary statement restores data sets (or the entire data base) using the backup file created by DBSTORE. Only the portion of the data base stored by DBSTORE are restored. Before executing DBRESTORE, a DBPURGE command should be executed to purge all data sets to be restored. If the root file was stored using DBSTORE, the root file must also be purged using DBPURGE. When executed from the keyboard, DBRESTORE displays either the set number of the set being restored or an * when the root file is being restored.

The volume specifier is used to specify the location of the root file. If the root file was stored using DBSTORE, DBRESTORE creates and restores the root file on the specified volume. If the root file was not stored, the volume specifier is used to specify the location of the existing root file. If no volume specifier is supplied, the default mass-storage device is used. Data sets stored by DBSTORE that were defined in the schema without a volume specifier are restored on the root file volume.

NOTE

When the root file is stored using DBSTORE, only those data sets stored with the root file are associated with the data base following a DBRESTORE. All other data sets are considered "uncreated" by IMAGE, and cannot be accessed. These data sets must be purged using a special mode of DBPURGE, and then created using DBCREATE before they may be used.

Backup and Recovery

IMAGE/250 software is designed to maintain and ensure the integrity of data bases. There is the possibility, however, of losing data or data base structural information due to hardware failure, an operating system error, or some other external cause. It is, therefore, prudent to adopt backup procedures of one type or another in anticipation of possible trouble. Two different approaches to data base backup and recovery are given below.

Method 1: DUPL

One approach to data base backup is to use the DUPL utility program daily to duplicate all discs containing updated data base files. If data is later lost due to a system failure, the backup disc may be copied back to the original disc by using DUPL. This backup approach is viable if:

- All frequently-used data bases are located on a small number of discs.
- Recovery from a midday system failure may be accomplished by re-running all jobs that modified the data base before the failure.

Method 2: DBSTORE

A second approach to data base backup is to backup each data base independently using DBSTORE. If data from a data base is then lost due to a system failure, it may be restored using DBRESTORE. This approach has the following advantages.

- Data bases may be backed up according to their frequency of use.
- ONLY data within the data base files are copied, which may take less time than a disc-to-disc duplicate method.

Each of these methods merely restores the data base to its state at backup time. No automatic recovery is provided to recover changes made between the time the backup copy of the data base is created and when the failure occurred. Providing backup can be difficult when many applications are concurrently modifying a single data base. In such cases, it may be necessary to backup the data base more frequently.

Data Base Utility Programs

The two utility programs, DBUNLD and DBLOAD, are used to copy data entries in data base restructuring operations, and in data recovery operations. These programs utilize a backup file (which may span several volumes) to store the data entries of all or selected data sets. The backup files used here are not compatible with the files created by DBSTORE, although both files are type BKUP. Both utility programs request data set volumes and backup volumes as needed. These programs require exclusive access to the data base (i.e., the data base cannot be open).

NOTE

Data bases having a combined global and local DBCB length greater than 4240 bytes may require more than 32k bytes of users memory for DBUNLD and DBLOAD to operate.

The DBUNLD Program

The DBUNLD utility program (data base unload) copies data set entries to a backup file. This program, located on the Utilities disc, is divided into two parts, having the program names DBUNLD and DBULD. The FORM files¹ DBFM1x and DBFM2x must appear on the same disc as the program files. An error file, UNERRx, is used but is not required for program operation. Error numbers and messages are listed in Appendix E.

To run the DBUNLD program, execute the command:

```
RUN" DBUNLD[volume spec]"
```

The volume specifier must appear when the DBUNLD program is not on the default mass-storage device. DBUNLD operates correctly regardless of which device contains the DBUNLD program files, and which device has been designated to be the default disc drive (using the MSI command).

¹ The last character is a revision code from A thru Z.

Once the RUN command has been executed, DBUNLD displays this form:

```

HP250.5.E          DATA BASE UNLOAD UTILITY
                   PARAMETER INPUT

Data Base Name [REDACTED]      Root File Volume Name [REDACTED]
Maintenance Password [REDACTED] Spool File Information (Optional)
                                File Name [REDACTED]
                                Volume Name [REDACTED]
Unload from Data BASE
CHAINED Mode Unload          Data Set Name [REDACTED]
Backup File Name [REDACTED]   Checkread ON
List of Backup Volume Names [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]

Please complete this form.

CHANGE SOURCE  CHANGE MODE  CLEAR FORM  CHANGE CHECKREAD  ACCEPT INPUT  EXIT PROGRAM
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
  
```

This information may now be entered in any order:

- Data Base Name** Name of the data base to be unloaded.
- Root File Volume Name** Blank response defaults to default disc.
- Maintenance Password** Defined by initial DBCREATE.
- Spool File Name** Name of the error message log file.
- Spool Volume Name** Volume to hold the error message log file.
- Data Set Name** Used when unloading an individual data set.
- Backup File Name** To be created by DBUNLD.
- List of Backup Volume Names** Volumes to contain backup file.

Press the CLEAR FORM key (or SFK 12) any time during form entry to erase all entries in the form. Press the EXIT PROGRAM key (or SFK 16) to terminate the program.

DBUNLD may be used to either unload an entire data base (all data set entries except automatic-master set entries) or a particular data set. The currently selected option, either unload data base or unload data set, is displayed on the form. When a single data set is to be unloaded (unload data set option), the data set name must be entered into the form. The data set name entry is ignored when the unload data base option is selected.

Data entries in detail data sets may be unloaded in either serial mode or chained mode (all master data sets are unloaded in serial mode). The currently-selected unload mode, either serial or chained, is displayed on the form. Press CHANGE MODE (or SFK 10) to change the selected unload mode.

In serial mode, detail data set entries are unloaded in physical order. This mode is somewhat faster than chained mode, since disc head movement is reduced. Data bases that have been marked corrupt by IMAGE must be unloaded in serial mode.¹ In addition, an attempt is made to recover as many entries as possible in a data set following a read data error (errors 87 and 88) on the data set.²

In chained mode, detail data set entries are unloaded along the primary path. This mode is somewhat slower than serial mode. The chained mode, used in conjunction with DBERASE and DBLOAD, is used to improve the access time for chained access along a detail data set's primary path. Entries can be unloaded and reloaded with entries in chained order, thus reducing disc head movement during chained access.

To log error messages while DBUNLD is running unattended, enter the name and volume of the spool file to which the errors should be logged. To use this feature, the SPOOL DROM must be loaded. Use the COPY command to list the contents of the log file after DBUNLD finishes. The COPY DROM must be loaded to execute this command.

CAUTION

Do not remove the volume containing the error log file during the unloading process. Error 142 will occur, and DBUNLD will halt.

¹ If chained mode is selected and the data base is marked corrupt, DBUNLD will issue an error message and terminate. Corrupt data bases must be unloaded serially since chain information may be missing or incorrect.

² If, following a read data error, DBUNLD detects that one or more entries have been lost due to the error, the unload process is terminated following the unloading of that data set. If the unload data base option was selected, any data sets not unloaded following the error may be unloaded using the unload data set option.

When all entries have been entered into the form, and all options and modes have been selected, press ACCEPT INPUT (or SFK 15) to begin processing. DBUNLD now checks that all required entries have been filled and displays:

DATA BASE UNLOAD UTILITY							
UNLOAD PROCESSING							
Data Number	Data Number	Data Number	Data Number	Data Number			
Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded			
Opening data base.							
							EXIT

Processing then continues, with DBUNLD displaying the data set number and number of entries unloaded. During processing, DBUNLD requests inserting data set volumes, backup volumes, and the volumes containing DBUNLD programs as needed. If an insufficient number of backup volumes is given, the program requests the names of additional backup volumes. Informational messages and requests for operator action are displayed on the line directly below the solid line as shown in the figure. Error messages are displayed on the bottom line.

To terminate the current operation, press EXIT (or SFK 16) anytime during program execution.

Following completion of the unload operation, or following an error or terminate operation, DBUNLD displays:

DATA BASE UNLOAD UTILITY UNLOAD PROCESSING	
Data Set	Number Unloaded
3	5
4	9
5	40
6	15

Please Select a function.

						RESTART	EXIT PROGRAM

Press EXIT PROGRAM (or SFK 16) to terminate the DBUNLD program. Press RESTART (or SFK 15) to restart the program. Following a restart DBUNLD displays the parameter input form, and displays all the data entered from the previous operation. Data may be either edited or cleared and re-entered as required.

The DBLOAD Program

The DBLOAD utility program loads data entries into a data base from a backup file created by DBUNLD. This program, located on the Utilities disc, is divided into three programs: DBLOAD, DBLOD and DBLD. The FORM files¹ DBFM3x, DBFM4x and DBFM5x must appear on the same disc as the program files. An error file, LDERRx, is used by DBLOAD to display error messages instead of error numbers, but is not required for program operation. A list of error codes and messages is in Appendix E.

To run the DBLOAD program, execute the command:

```
RUN"DBLOAD[volume spec]"
```

The volume specifier must appear when the DBLOAD program is not on the default mass-storage device. DBLOAD operates correctly regardless of which device contains the DBLOAD program files, and which device has been designated to be the default disc drive (using the MSI command).

Once the RUN command has been executed, DBLOAD displays:

HP250.5.E DATA BASE LOAD UTILITY
PARAMETER INPUT

Data Base Name <input type="text"/>	Root File Volume Name <input type="text"/>
Maintenance Password <input type="text"/>	Spool File Information (Optional)
Erase Data Base? (YES/NO) NO	File Name <input type="text"/>
Load into Data BASE	Volume Name <input type="text"/>
Backup File Name <input type="text"/>	Data Set Name <input type="text"/>
Re-order Items? (YES/NO) NO	Backup File Set Number <input type="text"/>
	First Backup Volume Name <input type="text"/>
	Checkread? (ON/OFF) ON

Please complete this form.

CHANGE EPASE	CHANGE DEST		CLEAR FORM	CHANGE CHECKREAD		ACCEPT INPUT	EXIT PROGRAM
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

¹ The last character is a revision code from A thru Z.

This information may now be entered in any order:

Data Base Name	Name of the data base to be loaded.
Root File Volume Name	Blank response defaults to default disc.
Maintenance Password	Defined by initial DBCREATE.
Spool File Name	Name of the error message log file.
Spool Volume Name	Volume to hold the error message log file.
Backup File Name	Name of file created by DBUNLD.
First Backup Volume Name	Location of first backup file segment.
Data Set Name	Used only when unloading single data set.
Backup File Set Number	Used only when unloading single data set.

Press CLEAR FORM (or SFK 12) any time during form entry to erase all entries in the form. Press EXIT PROGRAM (or SFK 16) to terminate the DBLOAD program.

The DBLOAD program can optionally erase the entire data base (using DBERASE) before loading entries. The erase option should be selected when loading a corrupt data base following a serial DBUNLD. An erase operation is not required if the data base is being restructured, and has just been created using DBCREATE. The current erase option is displayed on the form, and is changed by pressing CHANGE ERASE (or SFK 9).

DBLOAD may be used to load all entries stored in the backup file, or only those entries from a particular data set. The currently-selected option, either load data base or load data set, is displayed on the screen. Press CHANGE DEST (or SFK 10) to change the selected option. When a single set is loaded (load data set option), the name of the data set to be loaded must be entered. The number of the data set in the backup file whose entries are to be used must be entered if its set number is different than the number of the set to be loaded. The data set name and backup file set number entries are ignored when the load data base option is selected.

To log error messages while DBLOAD is running unattended, enter the name and volume of the spool file to which the errors should be logged. To use this feature, the SPOOL DROM must be loaded. Use the COPY command to list the contents of the log file after DBLOAD finishes. The COPY DROM must be loaded to execute this command.

CAUTION

Do not remove the volume containing the error log file during the loading process. Error 142 will occur, and DBLOAD will halt.

When all entries have been entered into the form, and all options have been selected, press ACCEPT INPUT (or SFK 15) to begin processing. DBLOAD then checks that all required entries have been filled, and displays:

DATA BASE LOAD UTIL:
LOAD PROCESSING

Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
Opening data base.									
									EXIT



Processing then continues, with DBLOAD displaying the data set number and number of entries loaded. During processing, DBLOAD prompts for inserting data set volumes, backup volumes, and the volume containing the DBLOAD and DBLOD programs as needed. Informational messages and requests for operator action are displayed on the line directly below the solid line, as shown in the figure. Error messages are displayed on the bottom line.

To terminate the current operation, press EXIT (or SFK 16) any time during program execution.

When ACCEPT INPUT is pressed, the source and destination item types are checked for compatibility before the load operation is performed. During the load operation, data conversions are performed as required. Compatible types are:

Source Item Type	Compatible Destination Item Types
I	I, S, L
S	I ² , S, L
L	I ² , S ² , L
X	X ¹

The sub-item count of source and destination items may be unequal. When the source sub-item count is greater than the destination sub-item count, only the first elements from the source array are used. When the source sub-item count is smaller than the destination sub-item count, the remaining elements in the destination item array are filled with spaces or zeros.

- ¹ When the string lengths are unequal, the source string is either padded with spaces or truncated, as required, before loading. If non-blank data is lost, a warning is issued.
- ² When a short or real source item has a greater range or precision than the destination item type permits, then the destination item is given the closest possible value to the source item value and a warning is issued.

Following completion of the load operation, or following an error or terminated operation, DBLOAD displays:

DATA BASE LOAD UTILITY
LOAD PROCESSING

Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
3	5								
4	9								
5	40								
6	15								

Please select a function.

						RESTART	EXIT PROGRAM
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Press EXIT PROGRAM (or SFK 16) to terminate the DBLOAD program. Press RESTART to restart the program. Following a restart, DBLOAD displays the parameter input form, and displays all the data entered from the previous operation. Data may be either edited or cleared and re-entered as required.

When the load data set option is selected, a third DBLOAD option, the re-order items option, may be selected by pressing CHANGE RE-ORDER (or SFK 11). This option is generally used when restructuring a data base, and allows re-defined data entries to be loaded using the data stored in the backup file. When this option is selected, and ACCEPT INPUT is pressed, this form is displayed:

DATA BASE LOAD UTILITY
PARAMETER INPUT

Item	Order	Item	Order	Item	Order	Item	Order	Item	Order	Item	Order
1	14	27	40	53	66	79	92	105	118		
2	15	28	41	54	67	80	93	106	119		
3	16	29	42	55	68	81	94	107	120		
4	17	30	43	56	69	82	95	108	121		
5	18	31	44	57	70	83	96	109	122		
6	19	32	45	58	71	84	97	110	123		
7	20	33	46	59	72	85	98	111	124		
8	21	34	47	60	73	86	99	112	125		
9	22	35	48	61	74	87	100	113	126		
10	23	36	49	62	75	88	101	114	127		
11	24	37	50	63	76	89	102	115			
12	25	38	51	64	77	90	103	116			
13	26	39	52	65	78	91	104	117			

Please enter new item order (data set item position,).

		FILL FORM	CLEAR FORM			ACCEPT INPUT	EXIT
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Press CLEAR FORM (or SFK 12) any time during form entry to erase all entries in the form. Press EXIT (or SFK 16) to terminate the operation.

The re-order item form specifies the new item order in terms of the item order of the entries stored on the backup file. Enter the backup file item position in the space next to each data item. Specifying a 0 or leaving the item position blank causes that item to be zeroed (or filled with spaces if the item is a string) when entries are loaded. An example re-order specification is shown below.

Data Set Item Position	Backup File Item Position
1	1
2	2
3 ¹	0
4	4
5 ¹	
6	7
7	6
.	
.	
.	

Salvaging Data

If data base or structural information has been lost (corrupt data base), and no backup copy of the data base is available, it may be possible to salvage most or all of the data by serially reading the data entries using DBUNLD, erasing the data base, and reloading the entries using DBLOAD. Corrupt data bases cannot be unloaded using DBUNLD in CHAINED mode, however, since discrepancies in the internal linkages may exist.

¹ The item in this position is either set to 0 or filled with space characters.

Data Base Restructuring

Certain changes to an existing structure can be made without having to transfer data from the old data base to the new one.

The DBMODS utility allows changing many structural items, as described on the next pages. More extensive changes are possible by first unloading the data base and recreating the root file. The general sequence is:

1. Run the DBUNLD utility program to copy all entries to a backup file.
2. Purge the old data base using DBPURGE.
3. Redefine the data base and use the EDITOR utility to modify the schema.
4. Run the Schema Processor to create the root file.
5. Use DBCREATE to create and initialize the new data sets.
6. Run the DBLOAD utility program to load data entries stored in the backup file into the new data base.

Following are examples of changes that can be made to the data base structure using the DBUNLD and DBLOAD utility programs. Some of the changes listed may require loading data sets on an individual basis, using the item re-ordering option provided by DBLOAD. Other structural changes, such as the addition or deletion of a data set, can often be accomplished using the technique previously described.

- Adding, changing or deleting passwords and user-class numbers.
- Changing data set read- and write-class lists.
- Increasing data set capacities.
- Adding new data item definitions.
- Removing data items not used as search items.
- Rearranging the item order of a data set entry.
- Changing the length of string items.¹
- Changing a data item or data set name and all references to it.
- Changing numeric type items to another numeric type.²

See Restructuring a Data Base in Chapter 5 for example operations using the DBLOAD and DBUNLD utilities.

¹ Certain changes cannot be made to strings used as search items. For example, decreasing the string length may cause a duplicate search item value in a master data set. Increasing the length of a search item value causes no problems.

² Numeric value conversions are performed between integer short and real values. The resulting item values are as close to the starting values as the destination item type permits.

The DBMODS Utility

The data base modification (DBMODS) utility allows making certain changes in the data base structure without the need to unload and load data stored in the data base. DBMODS can be used to modify data base passwords, user class accesses, item names, item format numbers, set names, set capacities, and data base volume names. These changes do not affect data stored in the data base. However, modifications made by DBMODS may require minor changes to any applications programs that access the data base.

The DBMODS utility maintains two modification counts associated with the root file. One count, the password modification count, is incremented when any passwords (including the maintenance work) are modified. A second count, the data base modification count, is incremented when any other changes are made to the root file. No changes are made to the modification counts unless the root file is actually modified by the program. These counts enable the user to detect unauthorized data base modifications made via the utility.

To run the DBMODS utility, insert the SYSTEM disc in a drive and execute:

```
RUN "DBMODS,SYSTEM"
```

The initial menu requests the data base name, root file volume specifier and the maintenance word (if used):

DBMODS
DATA BASE ENTRY FORM

DATA BASE NAME _____ ROOT FILE VOLUME _____

DATA BASE MAINTENANCE WORD _____

Please complete this form.

ACCEPT DATA							EXIT PROGRAM

After entering all items, the utility opens the data base and displays the menu shown next. An error message indicates if root file cannot be found or the data base is already open by another user. If an error occurs, either re-enter another data base name or press EXIT PROGRAM.

The main selection menu allows running any of the list and modification routines. Each is explained in the following pages. The example screens used here show information obtained from the SAD (sales analysis) data base on a volume named DEMO.

DBMODS
MAIN SELECTION MENU **DATA BASE: SAD**

PASSWORD - List and modify data base passwords.

ITEM - List and modify item names and format numbers.

SET - List and modify set names, volumes, capacities, and access.

VOLUME - List and modify data base volumes.

PRINT SCHEMA - Print schema definition of data base.

SELECT PRINTER - Select default printer for print screen functions.

RESTART - Run program again using another data base.

Please select a function.

DATA BASE MODIFICATION COUNT: 0 **PASSWORD MODIFICATION COUNT: 0**

PASSWORD	ITEM	SET	VOLUME	PRINT SCHEMA	SELECT PRINTER	RESTART	EXIT PROGRAM

List and Modify Passwords

Press the **PASSWORDS** softkey in the main selection menu to list and modify all data base passwords and to modify user-class access capabilities. The **PASSWORDS** menu displays all user passwords and the maintenance word.

DDBMS PASSWORDS		DATA BASE: SAD
1 _____	11 _____	21 _____
2 _____	12 _____	22 _____
3 SECRETARY	13 _____	23 _____
4 _____	14 _____	24 _____
5 _____	15 MANAGER	25 _____
6 _____	16 _____	26 _____
7 _____	17 _____	27 _____
8 _____	18 _____	28 _____
9 _____	19 _____	29 _____
10 SALESMAN	20 _____	30 _____
		31 _____

Maintenance word: _____

Please select a function.

MODIFY MAINT WRD	MODIFY PASSWORD	MODIFY ACCESS				PRINT SCREEN	EXIT
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

To modify the maintenance word, passwords, or user-class access, press the appropriate softkey and enter the requested items.

Press **PRINT SCREEN** to obtain a hard-copy output of the password list on the currently selected printer (see page 4-30).

Press **EXIT** to return to the previous menu.

List and Modify Items

Press the ITEMS softkey on the main selection menu to list and modify data item names and format numbers. The ITEMS menu displays all data item names and numbers in the data base. If more than 30 items are found, NEXT GROUP and PREVIOUS GROUP softkeys appear to allow reviewing all items.

DBMODS
ITEMS DATA BASE: SMD

1 ADDRESS (0)	11 PRICE (0)	_____ ()
2 CITY (0)	12 PRODUCT-NO (0)	_____ ()
3 COUNTRY (0)	13 PROD-DESC (0)	_____ ()
4 DATE (0)	14 REGION (0)	_____ ()
5 NAME (0)	15 REGION-DESC (0)	_____ ()
6 OPTION-DESC (0)	16 REGION-TYPE (0)	_____ ()
7 OPTION-PRICE (0)	17 SALESPERSON (0)	_____ ()
8 OPTION-TYPE (0)	18 SHIP-DATE (0)	_____ ()
9 ORDER-DATE (0)	19 STATE (0)	_____ ()
10 ORDER-NO (0)	20 ZIP-CODE (0)	_____ ()

Item name (format)

Please select a function.

		SELECT GROUP	MODIFY ITEM			PRINT SCREEN	EXIT

The SELECT GROUP softkey allows selecting alternate ways of displaying data items: by schema number order or by data set order. When items are listed in data set order, all items for a particular data set are listed in schema definition order. If more than 30 items have been selected, additional items may be displayed using the NEXT GROUP and PREVIOUS GROUP keys.

The MODIFY ITEM softkey allows changing item names and format numbers. Any item in the data base can be modified; it need not be currently displayed.

Press PRINT SCREEN to obtain a hard copy of the current item list on the currently selected printer (see page 4-30).

Press EXIT to return to the previous menu.

List and Modify Data Sets

Press the SET softkey on the main selection menu to list and modify data set names, volumes, capacities, and user-class access. The SETS menu displays the first 20 data set names, along with their associated volume names and capacities. If more than 20 sets are found, NEXT GROUP and PREVIOUS GROUP softkeys appear to allow reviewing all data sets.

The screenshot shows a terminal window titled "DBMODS SETS" with "DATA BASE: SAD" in the top right. The main area contains a list of data sets with columns for set number, name, volume, and capacity. The first six rows are: 1 DATE (51), 2 ORDER (101), 3 PRODUCT (11), 4 LOCATION (17), 5 OPTION (300), and 6 CUSTOMER (100). Below the list is a horizontal line with the text "Set name, volume (Capacity)" and "Please select a function." At the bottom, there are eight softkey buttons: SELECT GROUP, MODIFY ACCESS, MODIFY SET, PRINT SCREEN, and EXIT, with two empty buttons on the left.

Set No.	Set Name	Volume	Capacity
1	DATE	(51)	()
2	ORDER	(101)	()
3	PRODUCT	(11)	()
4	LOCATION	(17)	()
5	OPTION	(300)	()
6	CUSTOMER	(100)	()
		()	()
		()	()
		()	()
		()	()
		()	()
		()	()

Set name, volume (Capacity)

Please select a function.

		SELECT GROUP	MODIFY ACCESS	MODIFY SET		PRINT SCREEN	EXIT

The SELECT GROUP softkey allows selecting alternate ways data sets are listed. You can list sets in schema order (ascending set number order), by data set volume order, or by user-class number. When sets are listed in data set volume order, all sets stored on a particular volume are listed in schema definition order. When sets are listed by user-class number, all sets accessible by a particular user-class number are listed in schema definition order.

The MODiFY SET softkey allows changing data set names, set volume names, and set capacities. When a volume name is changed on a created data set, the entries in the existing data set are copied to the new volume. A data sets¹ capacity may be changed only if the data set is not created. Any data set in the data base can be modified regardless of which sets are currently displayed.

¹ A created set's capacity may be changed by using DBUNLD to copy existing entries of the set (and related detail entries if the set is a master) to a backup file, using DBPURGE to purge the set, and then using DBMODS to change the sets capacity. The modified set may then be created using DBCREATE, and reloaded using DBLOAD.

Volume Selection

Press the **VOLUME** softkey on the main selection menu to list and modify data base volume names.

DEMOS VOLUMES			DATA BASE: SAD
1 SALES	9 _____	17 _____	
2 _____	10 _____	18 _____	
3 _____	11 _____	19 _____	
4 _____	12 _____	20 _____	
5 _____	13 _____	21 _____	
6 _____	14 _____	22 _____	
7 _____	15 _____	23 _____	
8 _____	16 _____		

Root file volume: DEMO _____

Please select a function.

MODIFY VOLUME	MOVE ROOT					PRINT SCREEN	EXIT

The **MODIFY VOLUME** softkey allows changing volume names in a root file volume table. Changing a volume name in this table caused **IMAGE** to search the new volume for specific data sets.

Existing data sets residing on the volume with the old volume name will become “lost”, and cannot be accessed by **IMAGE** unless a **PRINT LABEL** statement is used to change the old volume name to the name specified in the volume table. Volume names cannot be changed to an existing volume name, i.e., volume names must be unique within the volume table.

The **MOVE ROOT** key allows moving the root file to a different volume. The root file is copied to the specified volume and the old root file is purged. Since existing data sets may appear on the old root file volume, a new entry is added to the volume table, allowing **IMAGE** to locate data sets that reside on the old root file volume.

Press **PRINT SCREEN** to obtain a hard copy of the current volume list on the currently selected printer (see page 4-30).

Press **EXIT** to return to the previous menu.

Print Schema Listing

Press the PRINT SCHEMA softkey on the main selection menu to generate a schema definition of the data base.

DIMODS
SCHEMA LISTER **DATA BASE: SMO**

PRINT SCHEMA - Print the schema definition of the data base to the printer.

COPY TO FILE - Copy the schema definition of the data base to a new or existing data file.

EXIT - Return to main selection menu.

Please select a function.

PRINT SCHEMA	PRINT TO FILE						EXIT

Press PRINT SCHEMA to obtain a hard copy of the schema listing on the currently set printer (see the next page).

Press PRINT TO FILE to copy the schema listing to a data file. If the specified type DATA file does not already exist, the utility creates a new file. After the schema definition is copied to a data file, the file can be used by the EDITOR and Schema Processor (SCHEMA) utilities.

Press EXIT to return to the main selection menu.

Select Printer

Press the SELECT PRINTER softkey on the main selection menu to specify the output device used for DBMODS print operations. The selection menu is:

DBMODS PRINTER SELECT		DATA BASE: SAD					
SELECT PRINTER - Select the default printer for print screen functions.							
SET LINES/PAGE - Set the number of lines printed per page.							
EXIT - Return to main selection menu.							
SELECTED PRINTER IS: 8				LINES PER PAGE: 21			
Please select a function							
SELECT PRINTER	SET LINES PER PAGE						EXIT
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Press SELECT PRINTER to change the default device used for DBMODS print operations. An error indicates if the selected device is not on line or not ready.

Press SET LINES PER PAGE to change the default page length. The range is from 20 through 200 lines per page.

Press EXIT to return to the main selection menu.

The DBPASS Utility

The DBPASS Utility is a binary program which provides four statements allowing access to user-level passwords and maintenance words. Each statement requires exclusive access to the data base (i.e., the data base cannot be open). DBPASS is available on operating system discs beginning with rev.2.D. To load the DBPASS binary, execute the statement:

```
LOAD BIN "DBPASS,SYSTEM"
```

The DBPASS Statement

The DBPASS statement allows changing the password for a stated user-class number.

```
DBPASS root file spec ; user-class number ; old password TO new password
```

The root file spec is a string expression containing the root file name and, optionally, its volume specifier. The user-class number is a numeric expression ranging from 1 through 31. The old password and new password parameters are string expressions from 0 through 8 characters in length and may be terminated by a space or semicolon. Longer strings are automatically truncated. The old password specified must match the corresponding password on the root file. For example, the statement:

```
DBPASS "LEDGER",1,"FRED" TO "FRIEDA"
```

changes the password FRED to FRIEDA for user class 1.

The DBMAINT Statement

The DBMAINT statement allows changing the maintenance password for a specified data base.

```
DBMAINT root file spec ; old word TO new word
```

The root file spec is a string expression containing the root file specifier and, optionally, its volume specifier. The old word and new word parameters are string expressions from 0 through 16 characters. Nulls, spaces and rubouts will be compressed out. The old word specified must match the current maintenance word for the data base. The maintenance word is established when the root file is created via the DBCREATE statement (see page 4-2). For example, the statement:

```
DBMAINT "LEDGER", "SECRET" TO "MANAGER"
```

changes the maintenance password for the LEDGER data base from SECRET to MANAGER.

The READ DBPASSWORD Statement

The READ DBPASSWORD statement reads all user passwords from the specified data base into a string array.

```
READ DBPASSWORD file spec ; maintenance word ; string array variable
```

The string array must have at least 31 elements, with a dimensioned length of at least eight characters long. Passwords are read into array elements in numerical order, beginning with the password for user-class number 1. If no password exists for a user-class number, that array element is filled with eight spaces. Here is a program sequence which reads and displays the user passwords from a root file named PAYROL.

```
10      OPTION BASE 1
20      DIM Pass$(31)(8)
30      READ DBPASSWORD "PAYROL","MANAGER";Pass$(*)
40      DISP "User-class Numbers and Passwords on the PAYROL Data Base."
50      DISP
60      FOR Num=1 TO 31
70          IF Pass$(Num)<>"          " THEN DISP Num;Pass$(Num)
80      NEXT Num
90      END
```

The WRITE DBPASSWORD Statement

The WRITE DBPASSWORD statement re-assigns all passwords in the specified root file with those in a specified string array.

```
WRITE DBPASSWORD root file spec ; maintenance word ; string array variable
```

The string array must contain at least 31 elements, with a dimensioned length of at least eight characters. The string in the first element becomes the password for user-class number 1; the second string becomes the password for user-class number 2; etc. Passwords are between 0 through 8 characters, and may be terminated by a space or semicolon. Longer strings are automatically truncated. If an element in the string array is null, or if the first eight characters are blanks, the corresponding user-class number is not assigned a password.

Here is a program which lists, and then allows the user to change, passwords for a specified data base.

```

10     OPTION BASE 1
20     ON HALT GOTO Exit
30     DIM Pass$(31),File$(13),Maint$(8)
40     INTEGER Num,Ucn
50     DISP "␣",SPA(24);" EDIT DATA BASE PASSWORDS "
60 Start: !
70     ON ERROR GOTO Traperr
80     CURSOR (1,20),UL(80),(1,21)      ! Draw prompt line.
90     INPUT "Enter root file name (and volume spec):";File$(1;13)
100    INPUT "%Enter maintenance password:";Maint$(1;8)
110    CURSOR (1,21)
120    DISP "%Reading passwords."
130    READ DBPASSWORD File$,Maint$;Pass$(*)
140 Disp_list: ! Display password list.
150    CURSOR (14,4)
160    DISP "User-class Numbers and Passwords on ";TRIM$(File$);" Data Base."
170    DISP
180    FOR Num=1 TO 28 STEP 4
190        DISP SPA(6);Num;Pass$(Num),Num+1;Pass$(Num+1),Num+2;Pass$(Num+2),Num+3;Pass$(Num+3)
200    NEXT Num
210    DISP SPA(6);29;Pass$(29),30;Pass$(30),31;Pass$(31)
220 Edit: !
230    CURSOR (1,21)
240    Ucn=0
250    INPUT "%Enter password number to edit:";Ucn
260    IF NOT Ucn THEN Done              ! Exit if no number entered.
270    IF (Ucn<1) OR (Ucn>31) THEN 240  ! Loop if ucn out of range.
280    EDIT "Edit password:",Pass$(Ucn);!
290    GOTO Disp_list                  ! Re-display password list.
300 Done: ! Write passwords on root file.
310    CURSOR (1,21)
320    DISP "%Writing passwords."
330    WRITE DBPASSWORD File$,Maint$;Pass$(*)
340 Exit: !
350    DISP "%End of program."
360    STOP
370 Traperr: !
380    BEEP
390    CURSOR (1,19)                    ! Position cursor above line.
400    DISP "␣"
410    SELECT ERRN
420    CASE 56
430        DISP "FILE NAME NOT FOUND. TRY AGAIN."
440    CASE 58
450        DISP "FILE NOT A ROOT FILE. TRY AGAIN."
460    CASE 80
470        DISP "DISC DRIVE DOOR OPEN. CLOSE DOOR AND TRY AGAIN."
480    CASE 81 TO 99
490        DISP "MASS STORAGE ERROR";ERRN;","
500        WAIT 2000
510        GOTO Exit
520    CASE 220
530        DISP "INCORRECT PASSWORD. TRY AGAIN."
540    CASE ELSE
550        DISP ERRM$                    ! Display any other error here.
560        WAIT 2000
570        GOTO Exit
580    END SELECT
590    WAIT 2000
600    DISP "%␣"
610    GOTO Start
620    END

```

The XCOPY Utility

The XCOPY utility is a binary program which provides the XCOPY statement, an enhanced version of the COPY statement. XCOPY allows copying any file types except SYST and DROM. To load the XCOPY binary, execute:

```
LOAD BIN "XCOPY,SYSTEM"
```

The XCOPY syntax is:

```
XCOPY source file spec, file type, protect code  
TO dest file spec [ ;REPLACE ]
```

The source and destination file specs are string expressions containing the file name and, optionally, the volume spec. The file type is a string expression containing the four-digit file type (DSET, ROOT, or BKUP). The protect code is a string expression containing the protect code or the two-digit set number. For example:

```
XCOPY "REF1", "DSET", "01" TO "REF1"
```

As with the COPY statement, XCOPY creates the specified file on the specified volume and copies the file. If the destination file already exists, specify ;REPLACE to copy the source file to the existing file. Error 851 occurs if the files are not compatible when ;REPLACE is used.

When copying data sets, the data base for the source and destination sets must not be open. Copying data sets without the corresponding root file and related data sets causes unexpected results when accessed.

CHAPTER 5

Example Operations



Introduction

This chapter contains examples of defining, using and maintaining an IMAGE/250 data base. One section describes the design and definition of an example data base used to store sales analysis data. The next section contains programs utilizing the data base manipulation statements to enter and retrieve data from the data base. A third section shows how the data base utilities are used for data base backup and restructuring.

Data Base Design

The next figure shows a customer order form for a fictitious company. The company owner has decided to design an IMAGE data base to be used to store information from the order form. Once the data has been stored in the data base, the owner wishes to generate various sales reports using the order data.

Demonstration Bicycle Company
ORDER FORM

Order Date _____ (MMYY) Order Number _____
Ship Date _____ (MMYY) Region _____
Salesperson _____

Name: _____
Address: _____
City _____ State _____
Zip Code _____ Country _____

Product: _____ Price: _____

Options:

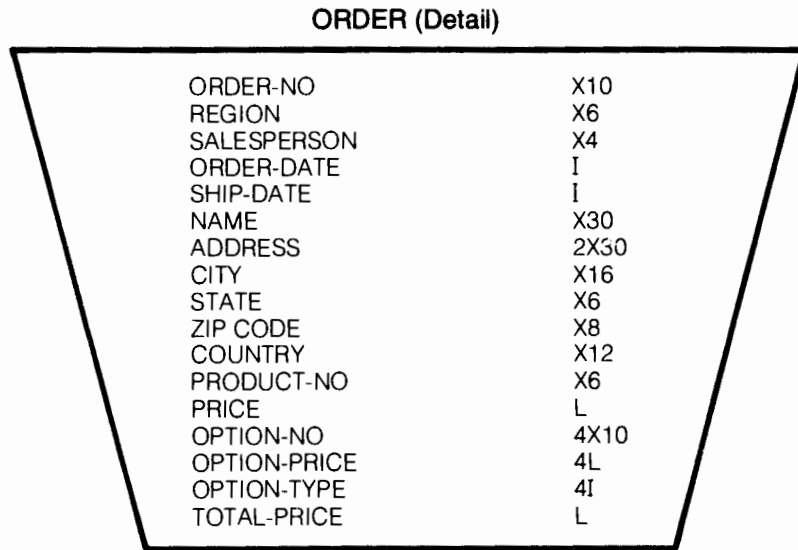
Option	Type	Price
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

Total: _____

Example Order Form

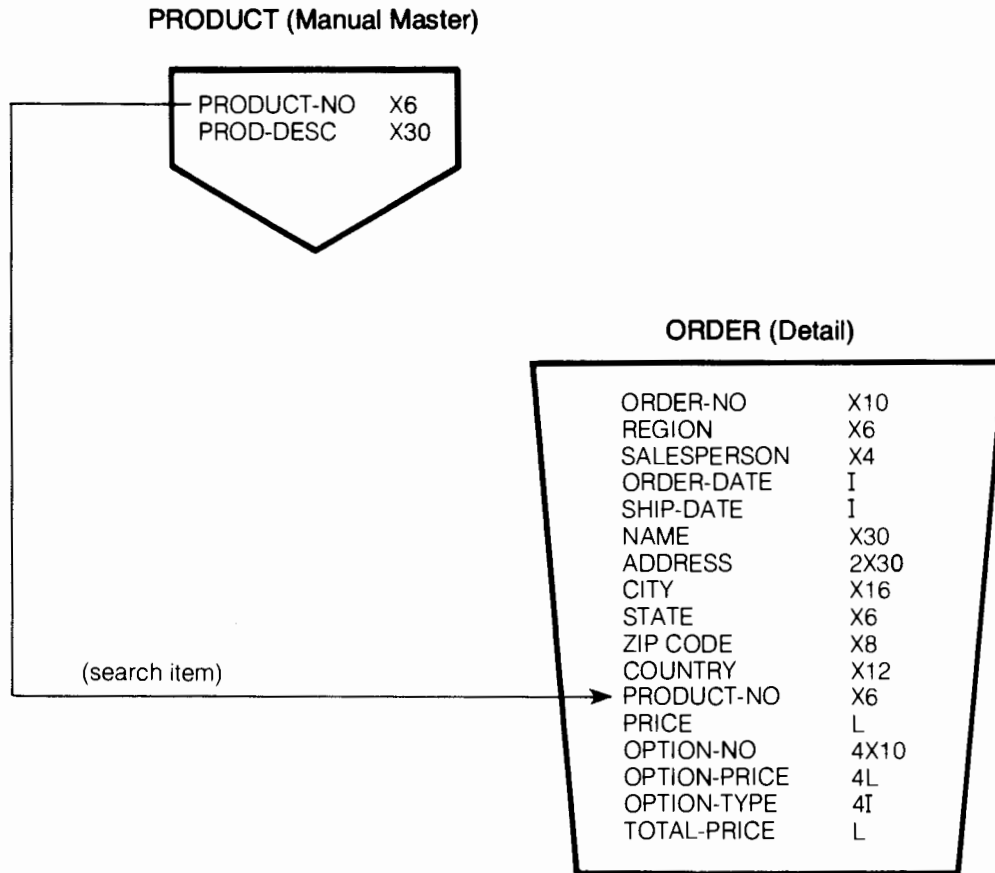
The data base structure shown next could be used to store the required data. This structure offers little advantage over using an HP 250 direct (random) access data file, but does provide a starting point for the data base design. Notice that in order to generate a list of all orders for a particular product, all entries in the detail set must be scanned. Also, no provision is made in this structure to handle an order which includes more than four options.

In order to provide a common basis for diagramming data base designs, pentagons are used to represent manual master data sets, triangles are used to represent automatic master data sets and trapezoids are used to represent detail data sets. The item names and item types used to define the data entry within the data set are also shown.



Possible Data Base Structure

The next data base structure shows a quick method of generating lists of orders for a particular product number. Using this structure, an applications program can perform a DBFIND (see Chapter 3) on the order detail data set to locate the chain head for a particular product number (PRODUCT-NO). The program can then perform successive chained DBGETs (see Chapter 3) to retrieve the required orders. Only the desired orders are accessed, thus reducing the time required to generate the list when a large number of orders are stored in the data base.



Possible Data Base Structure

Although an automatic master set could have been used for the PRODUCT master set, a manual master set was chosen for two reasons. First, since the PRODUCT set is a manual master, IMAGE/250 automatically checks the validity of the product number (PRODUCT-NO) as an order is entered into the data base. Only orders for products existing in the PRODUCT master set may be entered. Second, additional data may be stored in the PRODUCT data set, such as a product description (PRODUCT-DESC). Automatic master entries cannot contain items other than the search item.

The data base structure shown next utilizes a detail data set to store the option information, and a master data set to store the rest of the order data. This organization of data corresponds to dividing an order into two forms, as shown in the next figure. Unlike the previous structures shown, the number of options that may be stored with any order is limited only by the number of free entries in the OPTION data set. In addition, this organization requires less disc space to store order information when less than four options are ordered. (In the previous examples, the space to store four options with each order is allocated whether the options are purchased or not.) Notice that all order numbers within the ORDER data set must be unique since the order number is a search item.

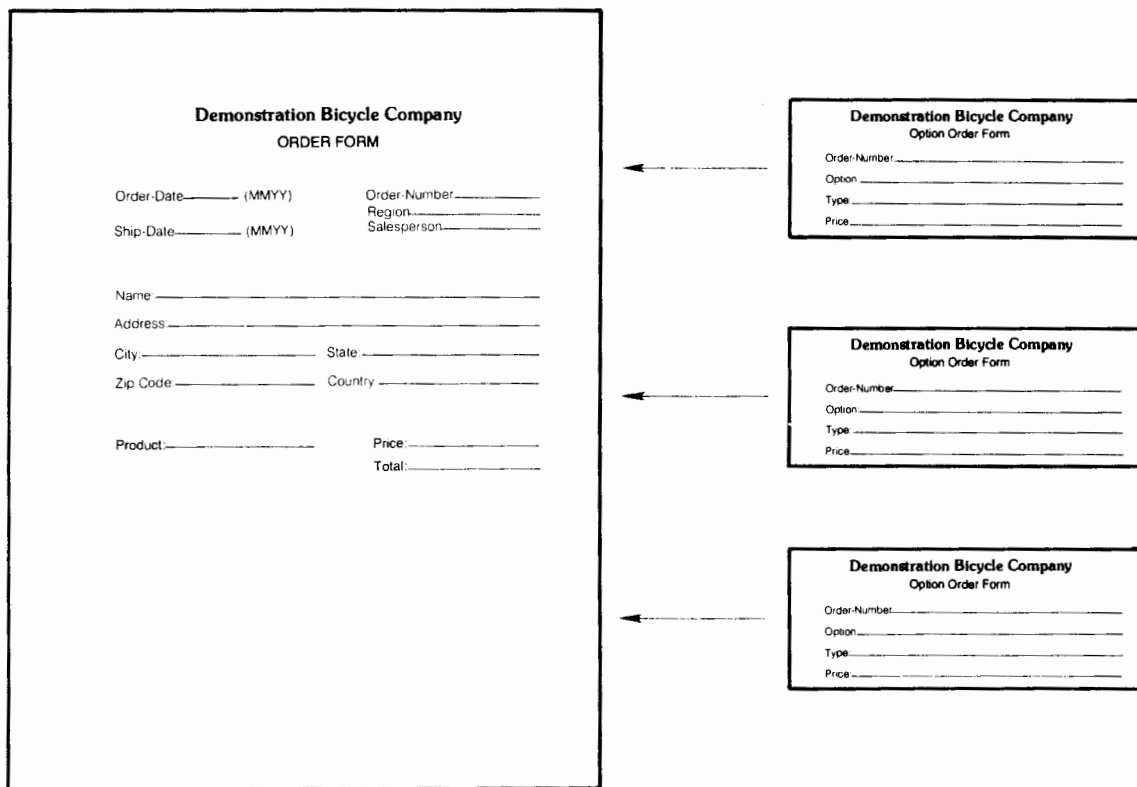
CUSTOMER (Manual Master)

ORDER-NO	X10
REGION	X6
SALESPERSON	X4
ORDER-DATE	I
SHIP-DATE	I
NAME	X30
ADDRESS	2X30
CITY	X16
STATE	X6
ZIP CODE	X8
COUNTRY	X12
PRODUCT-NO	X6
PRICE	L
TOTAL-PRICE	L

OPTION (Detail)

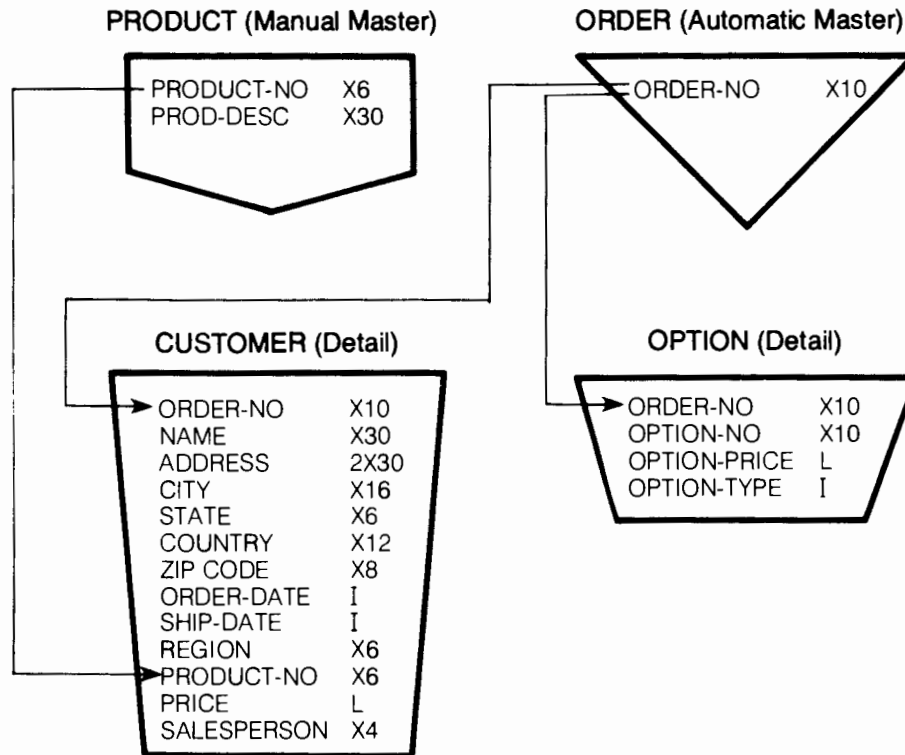
ORDER-NO	X10
OPTION-NO	X10
OPTION-PRICE	L
OPTION-TYPE	I

Possible Data Base Structure



Order Form with Separate Option Forms

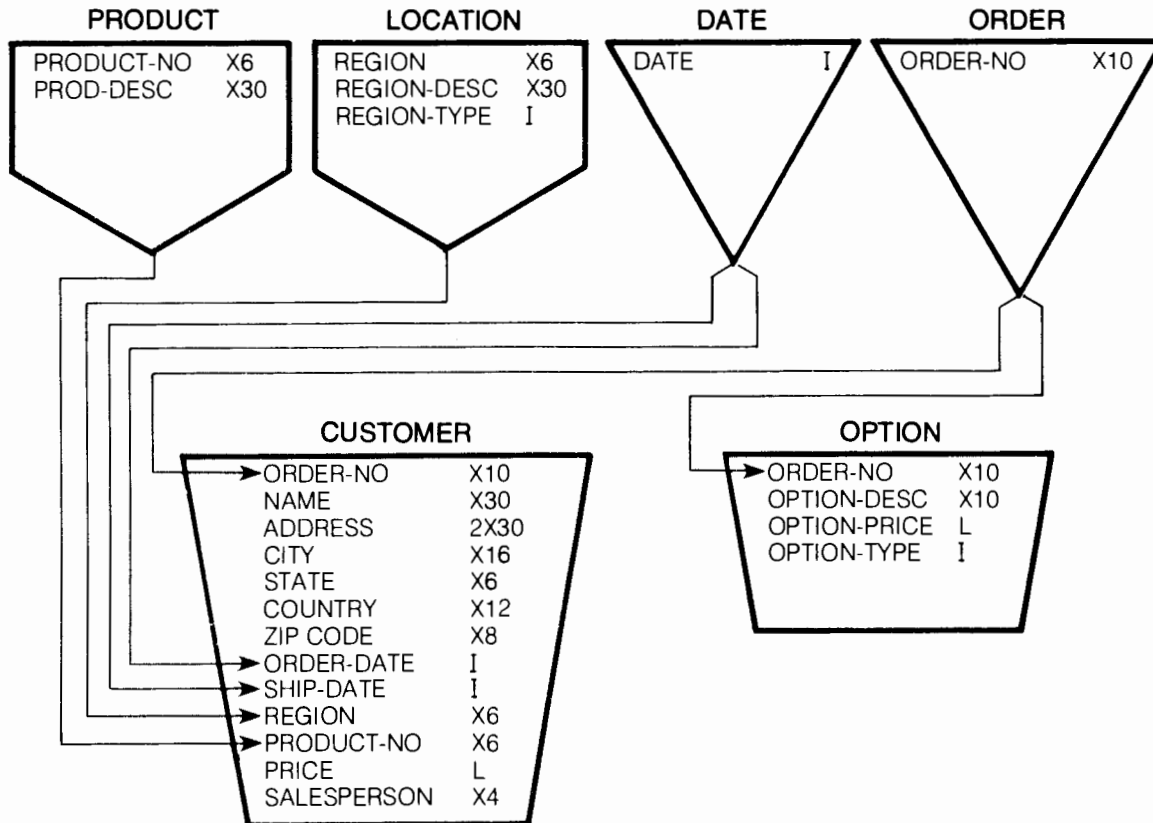
The advantages of the previous two structures are incorporated into the structure shown below. The CUSTOMER data set contains all order information except for option data, which is stored in the OPTION data set. Options are logically linked to the order through the ORDER master data set. Orders for a particular product are chained together using the PRODUCT master set.



Possible Data Base Structure

Although IMAGE/250 prevents entering orders for products not in the PRODUCT master, it does not prevent duplicate order numbers (ORDER-NO) from being entered. An order entry program may easily prevent the entry of duplicate order numbers, however, by performing a calculated DBGET on the ORDER master set before entering the order into the CUSTOMER and OPTION data sets.

The final data base design, which is used throughout the rest of this chapter, is shown below. A second manual master set, LOCATION, is used to chain orders from a particular sales region. A second automatic master, DATE, is used to locate orders for a particular order date or ship date. Dates are stored as an integer number (to reduce disc space requirements), and are easily converted to and from an ASCII character date within an applications program.



Sales Analysis Data Base

Data Base Definition and Creation

Once the data base has been designed, the data base definition language (DBDL) is used to define (describe) the data base. The EDITOR program, which is described in Appendix C, is used to create a data file containing this data base definition. The following listing, produced by the EDITOR program, shows the definition (schema) of the SAD data base using the DBDL. The KEEP command is then used to create a data file containing the definition. This file, named SADTXT, was created on volume SAD-VOL.

```

PAGE 1          HP250.1.A  EDITOR

1      #CONTROL      LIST, TABLE, ROOT
2      #TITLE "Sales Analysis Data Base"
3
4      BEGIN DATA BASE  SAD;  <<CUSTOMER SALES ANALYSIS DATA BASE>>
5
6      PASSWORDS:
7          10          SALESMAN;
8          15          MANAGER;
9          3           SECRETARY; <<WILL HAVE READ ACCESS ONLY>>
10
11     ITEMS:
12         ADDRESS,      2X30; <<2 LINES OF ADDRESS ALLOWED>>
13         CITY,         X16;
14         COUNTRY,     X12;
15         DATE,        I; <<PATH FOR ORDER-DATE, SHIP-DATE>>
16         NAME,        X30;
17         OPTION-DESC, X10;
18         OPTION-PRICE, L;
19         OPTION-TYPE, I;
20         ORDER-DATE,  I; <<MUST BE YYMM>>
21         ORDER-NO,    X10;
22         PRICE,       L;
23         PRODUCT-NO,  I;
24         PROD-DESC,   X30;
25         REGION,      X6;
26         REGION-DESC, X30;
27         REGION-TYPE, I;
28         SALESPERSON, X4;
29         SHIP-DATE,   I; <<MUST BE YYMM>>
30         STATE,       X6;
31         ZIP-CODE,    X8;
32
33     SETS:
34
35         NAME:        DATE, AUTOMATIC(3/10,15), SALES;
36         ENTRY:       DATE(2);
37         CAPACITY:    51;
38
39
40         NAME:        ORDER, A(3/10,15);
41         ENTRY:       ORDER-NO(2);
42         CAPACITY:    101;
43
44
45         NAME:        PRODUCT, MANUAL(3,10/15), SALES;
46         ENTRY:       PRODUCT-NO(1),
47                     PROD-DESC;
48         CAPACITY:    11;
49
50
51         NAME:        LOCATION, M(3,10/15), SALES;
52         ENTRY:       REGION(1),
53                     REGION-DESC,
54                     REGION-TYPE;
55         CAPACITY:    17;
56
57     #PAGE

```

(continued)

```

58      NAME:      OPTION,D(3/10,15);
59      ENTRY:     ORDER-NO(ORDER),
60                OPTION-DESC,
61                OPTION-PRICE,
62                OPTION-TYPE;
63      CAPACITY:  300;
64
65
66      NAME:      CUSTOMER,DETAIL(3/10,15);
67      ENTRY:     ORDER-NO(ORDER),
68                NAME,
69                ADDRESS,
70                CITY,
71                STATE,
72                COUNTRY,
73                ZIP-CODE,
74                ORDER-DATE(DATE),
75                SHIP-DATE(DATE),
76                REGION(LOCATION),
77                PRODUCT-NO(PRODUCT),
78                PRICE,
79                SALESPERSON;
80      CAPACITY:  100;
81
82      END.

```

After the data base definition is saved in the data file SADTXT, the Schema Processor (SCHEMA program) is used to create the root file for the SAD data base (see Chapter 2). For example, after executing

RUN "SCHEMA; SYSTEM"

the following is displayed:

SCHEMA PROCESSOR

Please enter the name of the schema text file.

CHANGE PRINTER								EXIT PROGRAM

The name and volume of the data file containing the data base definition are entered as shown below. The quotes around the file name and volume specifier are optional.

SCHEMA PROCESSOR

Please enter the name of the schema text file.
 "SADTXT,SAD_VOL" _

CHANGE PRINTER							EXIT PROGRAM

Here's the listing for the SAD data base, produced by the Schema Processor on the default printer:

```

PAGE 1      HEWLETT-PACKARD    HP250.1.A      IMAGE/250      Schema Processor

*CONTROL      LIST, TABLE, ROOT
*TITLE "Sales Analysis Data Base"

BEGIN DATA BASE  SAD;  <<CUSTOMER SALES ANALYSIS DATA BASE>>

PASSWORDS:
          10      SALESMAN;
          15      MANAGER;
           3      SECRETARY; <<WILL HAVE READ ACCESS ONLY>>

ITEMS:
          ADDRESS,      2X30; <<2 LINES OF ADDRESS ALLOWED>>
          CITY,         X16;
          COUNTRY,      X12;
          DATE,         I; <<PATH FOR ORDER-DATE, SHIP-DATE>>
          NAME,         X30;
          OPTION-DESC,  X10;
          OPTION-PRICE, L;
          OPTION-TYPE,  I;
          ORDER-DATE,   I;      <<MUST BE YYYYMM>>
          ORDER-NO,     X10;
          PRICE,        L;
          PRODUCT-NO,   I;
          PROD-DESC,    X30;
          REGION,       X6;
          REGION-DESC,  X30;
          REGION-TYPE,  I;
          SALESPERSON,  X4;
          SHIP-DATE,    I;      <<MUST BE YYYYMM>>
          STATE,        X6;
          ZIP-CODE,     X8;
    
```

(continued)

SETS:

NAME: DATE,AUTOMATIC(3/10,15),SALES;
 ENTRY: DATE(2);
 CAPACITY: 51;

NAME: ORDER,A(3/10,15);
 ENTRY: ORDER-NO(2);
 CAPACITY: 101;

NAME: PRODUCT,MANUAL(3,10/15),SALES;
 ENTRY: PRODUCT-NO(1),
 PROD-DESC;
 CAPACITY: 11;

NAME: LOCATION,M(3,10/15),SALES;
 ENTRY: REGION(1),
 REGION-DESC,
 REGION-TYPE;
 CAPACITY: 17;

PAGE 2

SAD

Sales Analysis Data Base

NAME: OPTION,D(3/10,15);
 ENTRY: ORDER-NO(ORDER),
 OPTION-DESC,
 OPTION-PRICE,
 OPTION-TYPE;
 CAPACITY: 300;

NAME: CUSTOMER,DETAIL(3/10,15);
 ENTRY: ORDER-NO(ORDER),
 NAME,
 ADDRESS,
 CITY,
 STATE,
 COUNTRY,
 ZIP-CODE,
 ORDER-DATE(DATE),
 SHIP-DATE(DATE),
 REGION(LOCATION),
 PRODUCT-NO(PRODUCT),
 PRICE,
 SALESPERSON;
 CAPACITY: 100;

END.

DATA SET NAME	TYPE	FLD CNT	PATH CNT	ENTRY LENGTH	MEDIA LENGTH	CAPACITY	SECTORS	VOLUME
DATE	A	1	2	2	20	51	4	SALES
ORDER	A	1	2	10	28	101	12	
PRODUCT	M	2	1	32	44	11	2	SALES
LOCATION	M	3	1	38	50	17	4	SALES
OPTION	D	4	1	30	34	300	40	
CUSTOMER	D	13	5	166	186	100	73	
							7	ROOT FILE LENGTH:
							142	TOTAL SECTORS INCLUDING ROOT:

NUMBER OF ERROR MESSAGES: 0
 ITEM NAME COUNT: 20 DATA SET COUNT: 6
 GLOBAL DBCB LENGTH: 1024 LOCAL DBCB LENGTH: 632

ROOT FILE SAD GENERATED.

Once the Schema Processor has created the root file, the data set files are created using DBCREATE (see Chapter 4). The command used to create the SAD data base is shown below. Both volumes SAD-VOL and SALES are on-line when the command is executed.

```
DBCREATE "SAD", "PASS"  
2 5 6 1 3 4
```


IMAGE/250 Programming Examples

Once a data base has been defined using the Schema Processor, and created using DBCREATE (see Chapter 2), data may be written to and read from the data base using the manipulation commands (Chapter 3). This section gives examples of each of the IMAGE/250 manipulation statements. All programs work with the Sales Analysis Data base (SAD) discussed in the previous section.

Example Program 1

One of the simplest reports that can be produced is a list of a data set's contents. A sample listing of the contents of the CUSTOMER data set is shown next, as produced by example program 1.

```
                                OUTSTANDING ORDERS LIST

ORDER NUMBER      CUSTOMER NAME                PRICE
-----
100                Smith, Thomas A.             175.50
101                Noname, Joseph              77.50
102                Johnson, Sam                162.50
103                Hernandez, Jose             109.75
104                Houseman, Sean             133.00
105                Sono, Jomo A.              135.00
106                Heining, Heinz             175.00
107                Dalling, Jimmy             150.00
108                Arauja, Luciano A.         80.00
109                Bekker, Bart               125.00
110                Gissing, Malcomb           45.00
                                =====
TOTAL ORDERS                        1368.25
```

For the programmer who has not used IMAGE/250, there are several small, but important, details which should be noted. In line 1080, B\$ is defined as the data base name. Two blanks must precede the name. The DBOPEN statement (line 1100) fills these blanks with a data base id number (two ASCII digits from 00 thru 04). This id number is used in subsequent DBML statements to identify the data base, rather than the data base name.

Note that the DBOPEN statement opens the data base for exclusive access. This means that if another user attempts to open the data base, an error occurs when DBOPEN is executed. This error takes the form of a non-zero value in the first element of the status array S(*). S(*) must be of type integer and must contain at least ten elements, in this case S(0) thru S(9).

```

1000 !   EXAMPLE PROGRAM 1
1010 !
1020 !   OUTSTANDING ORDERS REPORT (NOT INCLUDING ALL DETAIL)
1030 !
1040 !   INTEGER S(9)
1050 !   DIM B$(12),P$(10),Buf$(170)
1060 !   DIM Desc$(30),Order_no$(10),Name$(30)
1070 !   DISP "%c";           ! CLEAR SCREEN
1080 !   B$="  SAD,SALES"
1090 !   P$="MANAGER"
1100 !   DBOPEN (B$,P$,3,S(*)) ! OPEN FOR EXCLUSIVE ACCESS
1110 !   IF S(0) THEN Dberr
1120 !
1130 !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1140 !
1150 !   Rep:Total=0
1160 !   Eof=11
1170 !   PRINT TAB(20);"OUTSTANDING ORDERS LIST";LIN(1)
1180 !   PRINT "ORDER NUMBER  CUSTOMER NAME";SPA(14);"PRICE";
      !   LIN(1);RPT$("- ",48);LIN(1)
1190 !
1200 !   PRODUCE THE REPORT
1210 !
1220 !   Start_report:DBGET (B$,"CUSTOMER",2,S(*),"@",Buf$,0)
1230 !   IF S(0)=Eof THEN End_report
1240 !   IF S(0) THEN Dberr
1250 !   UNPACK USING Pf2;Buf$
1260 !   Pf2:PACKFMT Order_no$,Name$,60X,16X,6X,12X,8X,2X,2X,6X,2X,
      !   Price
1270 !   PRINT USING Itm_image;Order_no$,Name$,Price
1280 !   Itm_image:IMAGE 16A,22A,2X,5D.DD
1290 !
1300 !   ACCUMULATE TOTAL
1310 !
1320 !   Total=Total+Price
1330 !   GOTO Start_report
1340 !
1350 !   PRINT FINAL TOTALS
1360 !
1370 !   End_report:PRINT USING Tot_image;Total
1380 !   END
1390 !   Tot_image:IMAGE 39X,9("=") / 3X,"TOTAL ORDERS",24X,6D.DD /
1400 !
1410 !   DATA BASE ERROR HANDLER
1420 !
1430 !   Dberr:DISP LIN(1);"UNEXPECTED DATA BASE ERROR ";
      !   VAL$(S(0));" IN LINE";S(6)
1440 !   END

```

The status array should be checked for an abnormal condition (non-zero first element) after each DBML operation. If an abnormal condition is detected in this program, control is transferred to the line labeled Dberr, which displays the error code in the status array and the line number where the error occurred.

The process of reading all orders is accomplished by a loop containing a serial-access DBGET (see line 1220). This line reads the next non-empty record from the CUSTOMER set and puts the record into the string Buf\$.

The pertinent items from this buffer are extracted via the UNPACK USING statement and then printed. Note the use of Xs to skip unused fields in the PACKFMT. For clarity, each X field corresponds to a field in the data base. The whole group could be replaced, however, by 114X. Before reading the next record, the price of the current order is added into the total price of all orders read so far.

When the orders in the CUSTOMER set are exhausted, the first element of the status array, S(0), indicates when an end-of-file has been reached. Line 1230 detects this and branches to print the total of all order prices.

Example Program 2

Example program 2 prints a list of all orders grouped by product. This is accomplished by serially reading each entry in the PRODUCT set (see line 1220). Then, for each product, a listing of the record contents with the same product number in the CUSTOMER detail data set is produced.

It is not necessary to scan the entire CUSTOMER section to find entries with the correct product number. The chain between the CUSTOMER set and the PRODUCT set with PRODUCT-NO as the search field allows direct access to those entries in the CUSTOMER set with a particular product number. To access the entries on the chain, a DBFIND is first executed (line 1340). Status array element S(5) returns the number of entries in the chain. A FOR-NEXT loop going from 1 to S(5) with a chained mode DBGET (line 1370) extracts the information for each order with the desired product number.

The procedure reads the chain in a "forward" direction. It is possible to read the chain backwards by using a direct mode DBGET and chain pointers which are returned in the status array by both DBFIND and DBGET. To change example program 2 to do a backward chain read, line 1370 is replaced by:

```
1370 DBGET (B$, "CUSTOMER", 4, S(*), "@", Buf$, S(7))
```

Example program 2 also solves one of the problems of program 1: by opening the data base in mode 8 (read-only mode) instead of mode 3 (exclusive access), other users may also open the data base in mode 8 and do concurrent reads. Opening in mode 8, however, fails if another user has the data base opened in either mode 3 or mode 1. As long as the data base is opened by one user in mode 8, no one can open it in a mode which permits modifications to the data base.

OUTSTANDING ORDERS LIST			
PRODUCT	ORDER NUMBER	CUSTOMER NAME	PRICE

50 (Tricycle)			
	110	Gissing, Malcomb	45.00
			=====
	TOTAL ORDERS FOR 50		45.00
1000 (10-Speed Bicycle)			
	102	Johnson, Sam	162.50
	106	Heining, Heinz	175.00
	107	Dalling, Jimmy	150.00
			=====
	TOTAL ORDERS FOR 1000		487.50
100 (Standard Bicycle)			
	101	Noname, Joseph	77.50
	103	Hernandes, Jose	109.75
	108	Arauja, Luciano A.	80.00
			=====
	TOTAL ORDERS FOR 100		267.25
300 (3-Speed Bicycle)			
	104	Houseman, Sean	133.00
			=====
	TOTAL ORDERS FOR 300		133.00
500 (5-Speed Bicycle)			
	100	Smith, Thomas A.	175.50
	105	Sono, Jomo A.	135.00
	109	Bekker, Bart	125.00
			=====
	TOTAL ORDERS FOR 500		435.50
		TOTAL ORDERS	\$1368.25
			=====

```

1000 !   EXAMPLE PROGRAM 2
1010 !
1020 !   OUTSTANDING ORDERS REPORT (NOT INCLUDING ALL DETAIL)
1030 !
1040   INTEGER S(9),Prod_no
1050   DIM B$(12),P$(10),Buf$(170)
1060   DIM Desc$(30),Order_no$(10),Name$(30)
1070   DISP "*c";           ! CLEAR SCREEN
1080   B$="  SAD,SALES"
1090   P$="MANAGER"
1100   DROPN (B$,P$,8,S(*)) ! OPEN FOR READ-ONLY ACCESS
1110   IF S(0) THEN Dberr
1120 !
1130 !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1140 !
1150 Rep:Total=Master_total=0
1160   Eof=11
1170   PRINT TAB(20);"OUTSTANDING ORDERS LIST";LIN(1)
1180   PRINT "PRODUCT      ORDER NUMBER  CUSTOMER NAME";
      SPA(14);"PRICE";LIN(1);RPT$("- ",63);LIN(1)
1190 !
1200 !   PRODUCE THE REPORT
1210 !
1220 Start_report:DBGET (B$,"PRODUCT",2,S(*),"@",Buf$,0)
1230   IF S(0)=Eof THEN End_report
1240   IF S(0) THEN Dberr
1250   UNPACK USING Pf;Buf$
1260 Pf:  PACKFMT Prod_no,Desc$
1270 !
1280 !   PRINT HEADER FOR PRODUCT
1290 !
1300   PRINT VAL$(Prod_no);" (";TRIM$(Desc$);")"
1310 !
1320 !   PRINT ORDERS
1330 !
1340   DBFIND (B$,"CUSTOMER",1,S(*),"PRODUCT-NO",Prod_no)
1350   IF S(0) THEN Dberr
1360   FOR I=1 TO S(5)
1370     DBGET (B$,"CUSTOMER",5,S(*),"@",Buf$,0)
1380     IF S(0) THEN Dberr
1390     UNPACK USING Pf2;Buf$
1400 Pf2:  PACKFMT Order_no$,Name$,60X,16X,6X,12X,8X,2X,2X,6X,
      2X,Price
1410     PRINT TAB(16);
1420     PRINT USING Itm_image;Order_no$,Name$,Price
1430 Itm_image:IMAGE 16A,22A,2X,5D.DD
1440 !
1450 !   ACCUMULATE TOTALS
1460 !
1470     Total=Total+Price
1480     Master_total=Master_total+Price
1490   NEXT I
1500 !
1510 !   PRINT TRAILER FOR PRODUCT
1520 !
1530   PRINT TAB(54);
1540   PRINT USING Tot_image;VAL$(Prod_no),Total
1550   Total=0
1560   GOTO Start_report
1570 !
1580 !   PRINT FINAL TOTALS
1590 !
1600 End_report:PRINT USING Mstr_image;Master_total
1610   END
1620 Tot_image:IMAGE 9("=") / 3X,"TOTAL ORDERS FOR ",10A,24X,
      6D.DD /
1630 Mstr_image:IMAGE // 25X,"TOTAL ORDERS",14X,"$"9D.DD / 54X,
      9("=")
1640 !
1650 !   DATA BASE ERROR HANDLER
1660 !
1670 Dberr:DISP LIN(1);"UNEXPECTED DATA BASE ERROR ";
      VAL$(S(0));" IN LINE";S(6)
1680   END

```

Example Program 3

Example program 3 allows other users to perform write operations (DBPUT, DBDELETE and DBUPDATE) by opening the data base in mode 1. Other users can now open the data base in mode 1 for reading. By using the locking capability, other programs can perform puts, deletes and updates.

This program is basically an expansion of the previous example. The options for each order are listed by inserting a chained-access DBGET through the OPTION set for each order found in CUSTOMER. The DBFIND on the OPTION set (line 1470) using the order number from CUSTOMER finds the head of the chain of options. The FOR-NEXT loop (line 1490) then chains through the entries in the OPTION set, doing chained-mode DBGETS.

A significant feature of this program is the replacement of the UNPACK statements with IN DATA SETs. Lines 1180 thru 1200 set up a correspondence between variables in the BASIC program and fields in the data sets. Thus, when the DBGET on PRODUCT is performed (line 1310), the values of the fields PRODUCT-NO and PRODUCT-DESC in the PRODUCT set are automatically assigned to the variables Prod-no and Desc\$. Similarly, when the DBGET in line 1500 is executed, new values are assigned to Option-desc\$ and P0.

The use of SKP in the IN DATA SET for the OPTION set (line 1200) instructs the system to ignore the value of the ORDER-NO field. It is not assigned to any variable since this field was read by the DBGET on the CUSTOMER set and assigned to Order-no\$; reassigning it would be superfluous.

The USE ALL option on the IN DATA SET for the CUSTOMER set (line 1190) specifies that only fields whose names correspond to variables already in the program are to be unpacked into their corresponding variables.

Only the variables Order-no\$ and Name\$ in this program correspond to fields in the CUSTOMER set. Thus, when the DBGET in line 1410 is executed, only the value of the fields ORDER-NO and NAME are assigned to variables (Order-no\$ and Name\$, respectively).

Another feature of example program 3 is error control and program termination. Lines 1080 and 1090 allow the program to trap the HALT key and any error conditions and wrap-up gracefully. Both termination conditions, as well as the data base routine, then attempt to close the data base (line 1900). In this instance, however, the DBCLOSE is not critical. Had any write operations been performed, the close would be necessary to properly record the changes made to the data base.¹

¹ A mode 4 DBCLOSE is automatically executed when the program ENDS when the data base is not closed prior to program completion. This close updates the data stored on the disc, but leaves the data base open for further access. Certain operations, such as DBERASE, require exclusive access to the data base, and cannot be performed until a mode 1 DBCLOSE is executed.

OUTSTANDING ORDERS LIST

PRODUCT	ORDER NUMBER	CUSTOMER NAME	OPTIONS	PRICE
50 (Tricycle)	110	Gissing, Malcomb		45.00
				45.00
TOTAL ORDERS FOR 50				45.00
1000 (10-Speed Bicycle)	102	Johnson, Sam	Chrome	150.00 12.50
				162.50
	106	Heining, Heinz	Light Basket	150.00 10.00 15.00
				175.00
	107	Dalling, Jimmy		150.00
TOTAL ORDERS FOR 1000				150.00 ===== 487.50
100 (Standard Bicycle)	101	Noname, Joseph	Horn	75.00 2.50
				77.50
	103	Hernandes, Jose	Light Mud Flaps Horn Stripes Fan	75.00 5.00 7.25 10.00 2.50 10.00
				109.75
	108	Arauja, Luciano A.	Horn	75.00 5.00
TOTAL ORDERS FOR 100				80.00 ===== 267.25
300 (3-Speed Bicycle)	104	Houseman, Sean	Light Super tire	110.00 5.00 18.00
TOTAL ORDERS FOR 300				133.00 ===== 133.00
500 (5-Speed Bicycle)	100	Smith, Thomas A.	Light BASKETLE	125.00 5.00 45.50
				175.50
	105	Sono, Jomo A.	Horn Reflector	125.00 2.50 7.50
				135.00
	109	Bekker, Bart		125.00
TOTAL ORDERS FOR 500				125.00 ===== 435.50
TOTAL ORDERS				\$1368.25 =====



```

1000 !   EXAMPLE PROGRAM 3
1010 !
1020 !   OUTSTANDING ORDERS REPORT (INCLUDING ALL DETAIL)
1030 !
1040 !   INTEGER S(9),Product_no,Prod_no
1050 !   DIM B$(12),P$(10),Buf$(170)
1060 !   DIM Desc$(30),Order_no$(30),Name$(30),
      !       Option_desc$(10)
1070 !   DISP "*c";           ! CLEAR SCREEN
1080 !   ON ERROR GOTO Error  ! SET UP ERROR AND HALT TRAPS
1090 !   ON HALT GOTO Halt
1100 !   B$="  SAD,SALES"
1110 !   P$="MANAGER"
1120 !   DBOPEN (B$,P$,1,S(9)) ! OPEN FOR SHARED ACCESS
1130 !   IF S(0) THEN Dberr
1140 !
1150 !   SET UP ALL APPROPRIATE RELATIONSHIPS
1160 !
1170 !   DBASE IS B$
1180 !   IN DATA SET "PRODUCT" USE Prod_no,Desc$
1190 !   IN DATA SET "CUSTOMER" USE ALL
1200 !   IN DATA SET "OPTION" USE SKP 1,Option_desc$,P0
1210 !
1220 !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1230 !
1240 !   Rep:Total=Master_total=0
1250 !   Eof=11
1260 !   PRINT TAB(30);"OUTSTANDING ORDERS LIST";LIN(1)
1270 !   PRINT "PRODUCT";SPA(8);"ORDER NUMBER";SPA(10);
      !       "CUSTOMER NAME";SPA(9);"OPTIONS";SPA(8);"PRICE";
      !       LIN(1);RPT$("- ",79);LIN(1)
1280 !
1290 !   PRODUCE THE REPORT
1300 !
1310 !   Start_report:DBGET (B$,"PRODUCT",2,S(9),"@",Buf$,0)
1320 !   IF S(0)=Eof THEN End_report
1330 !   IF S(0) THEN Dberr
1340 !
1350 !   PRINT HEADER FOR PRODUCT
1360 !
1370 !   PRINT VAL$(Prod_no);" (";TRIM$(Desc$);")"
1380 !   DBFIND (B$,"CUSTOMER",1,S(9),"PRODUCT-NO",Prod_no)
1390 !   IF S(0) THEN Dberr
1400 !   FOR I=1 TO S(5)
1410 !     DBGET (B$,"CUSTOMER",5,S(9),"@",Buf$,0)
1420 !     IF S(0) THEN Dberr
1430 !
1440 !     PRINT HEADER FOR ORDER
1450 !
1460 !     PRINT TAB(20);Order_no$;TAB(38);Name$(1,21);
1470 !     DBFIND (B$,"OPTION",1,S(9),"ORDER-NO",Order_no$)
1480 !     IF S(0) THEN Dberr
1490 !     FOR J=1 TO S(5)
1500 !       DBGET (B$,"OPTION",5,S(9),"@",Buf$,0)
1510 !       IF S(0) THEN Dberr
1520 !
1530 !       PRINT OPTIONS
1540 !
1550 !       PRINT TAB(60);
1560 !       PRINT USING Itm_image;Option_desc$,P0
1570 !   Itm_image:IMAGE 10A,2X,5D.DD
1580 !
1590 !     ACCUMULATE TOTALS
1600 !
1610 !     Total=Total+P0
1620 !     Sub_total=Sub_total+P0
1630 !     Master_total=Master_total+P0
1640 !   NEXT J
1650 !   PRINT TAB(71);
1660 !   PRINT USING Sub_image;Sub_total
1670 !   Sub_total=0
1680 ! NEXT I
1690 !
1700 !   PRINT TRAILER FOR PRODUCT
1710 !
1720 !   PRINT TAB(70);
1730 !   PRINT USING Tot_image;VAL$(Prod_no),Total
1740 !   Total=0
1750 !   GOTO Start_report

```

(continued)

```

1760 !
1770 !   PRINT FINAL TOTALS
1780 !
1790 End_report:PRINT USING Mstr_image;Master_total
1800   GOTO Close
1810 Sub_image:IMAGE 8("-") / 71X,5D.DD /
1820 Tot_image:IMAGE 9("=") / 11X,"TOTAL ORDERS FOR ",10A,32 X,
        6D.DD /
1830 Mstr_image:IMAGE // 31X,"TOTAL ORDERS",24X,"$"8D.DD / 70X,
        9("=")
1840 !
1850 !   ERROR AND HALT TERMINATION ROUTINES
1860 !
1870 Dberr:DISP LIN(2);"STATUS ERROR ";VAL$(S(0));" IN LINE";
        S(6)
1880   GOTO Close
1890 Error:DISP LIN(2);"UNEXPECTED ";ERRM$
1900   GOTO Close
1910 Halt:PRINT LIN(2)
1920 Close:DBCLOSE (B$,"",1,S(*))
1930   DISP LIN(2);"END OF OUSTANDING ORDERS REPORT."
1940   END

```


Example Program 4

The last two programs are used to enter new products into the data base and make modifications and deletions to existing products. Example program 4 allows new products to be added to the data base. Since write operations must be performed, the data base is opened in mode 3 (see line 1130). This program also contains the necessary lines to trap HALTs and errors (see lines 1090 and 1100).

When the program is first RUN it produces a screen like that shown below.

```
ADD NEW PRODUCT

Enter the number of the product you wish to add.
-

CURRENT NUMBER OF ENTRIES: 5

SET CAPACITY:          11

-----
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [EXIT PROGRAM]
```

Press EXIT PROGRAM to terminate the program. Otherwise, a new product number is entered and the program prompts for a product description. Note that the set capacity and the current number of entries are displayed. This information is obtained via DBINFO in line 1330. Note also, that since the result of DBINFO is left in the buffer, an UNPACK is used (line 1350) to extract these values into integer variables. Since integers range from -32768 thru 32767 and capacities range from 1 thru 65534, a conversion is necessary. The FNCorrect function defined in line 1240 provides this conversion. Example program 2, which does a direct mode access to perform a backward chain read (see page 2-14), also requires the use of this type of conversion since the capacity of the CUSTOMER set can exceed 32767.

If a duplicate product number is entered, the program displays an error and reprompts for a correction. For example:

ADD NEW PRODUCT							
Enter the number of the product you wish to add.							
50							
PRODUCT ALREADY IN DATA BASE.							
CURRENT NUMBER OF ENTRIES: 5							
SET CAPACITY: 11							
							EXIT
							PROGRAM

This duplicity is detected by attempting a calculated access (line 1460) of the master for the product using the given product number. If the specified product is not found, the product description is requested as shown next:

ADD NEW PRODUCT							
Enter the number of the product you wish to add.							
1							
Enter product description							
Uni-cycle_							
							EXIT
							PROGRAM

After the description is entered, a DBPUT (line 1600) is used to store the new product in the data base. The program then reports that the DBPUT was successful and gives two options which are key selected. The user may either add ANOTHER product or EXIT the program:

ADD NEW PRODUCT

Enter the number of the product you wish to add.
1

Enter product description
Uni-cycle
NEW PRODUCT ADDED.

ANOTHER							EXIT PROGRAM

```

1000 ! EXAMPLE PROGRAM 4
1010 !
1020 ! ADD NEW PRODUCT
1030 !
1040 ! INTEGER S(9),Prod_no,Entries,Capacity
1050 ! DIM B$(12),P$(10),Buf$(170)
1060 ! DIM Desc$(30)
1070 ! DISP "c"; ! CLEAR SCREEN
1080 ! DISP TAB(32);"ADD NEW PRODUCT"
1090 ! ON ERROR GOTO Error ! SET UP ERROR AND HALT TRAPS
1100 ! ON HALT GOTO Halt
1110 ! B$=" SAD,SALES"
1120 ! P$="MANAGER"
1130 ! DBOpen (B$,P$,3,S(*)) ! OPEN FOR EXCLUSIVE ACCESS
1140 ! IF S(0) THEN Dberr
1150 !
1160 ! SET UP ALL APPROPRIATE RELATIONSHIPS
1170 !
1180 ! DRASE IS B$
1190 ! IN DATA SET "PRODUCT" USE Prod_no,Desc$
1200 !
1210 ! FUNCTION TO CONVERT TO AN INTEGER FROM 1 TO 65534
1220 ! RATHER THAN FROM 1 TO 32767 AND -32768 TO -2
1230 !
1240 ! DEF FNCorrect(INTEGER N)=N+65536*(N<0)
1250 !
1260 ! INITIALIZE
1270 !
1280 ! Not_found=17
1290 ! ON KEY #8:"EXIT" GOTO Halt
1300 ! ON KEY #16 GOTO Halt
1310 ! Cont:DISP " ";LIN(3);"c";LIN(10)
1320 ! OFF KEY #1,9
1330 ! DBINFO (B$,"PRODUCT",202,S(*),Buf$)
1340 ! UNPACK USING Fmt;Buf$
1350 ! Fmt:PACKFMT 28X,Entries,2X,Capacity
1360 ! DISP "CURRENT NUMBER OF ENTRIES: ";FNCorrect(Entries);
! LIN(2)
1370 ! DISP "SET CAPACITY: ";FNCorrect(Capacity)

```

```

1380 !
1390 ! ASK FOR NEW PRODUCT NUMBER
1400 !
1410 Again:DISP "*" ;LIN(3);
1420 Bdp=-1 ! ALLOW FOR A NULL USER RESPONSE.
1430 INPUT "Enter the number of the product you wish to add.",
      Prod_no
1440 IF Prod_no(1 THEN Bdp
1450 DBGET (B$, "PRODUCT", 7, S(*), "@", Buf$, Prod_no)
1460 IF S(0)=Not_found THEN Enter
1470 IF S(0) THEN Dberr
1480 DISP "PRODUCT ALREADY IN DATA BASE."
1490 BEEP
1500 GOTO Again
1510 Bdp:DISP "ILLEGAL PRODUCT NUMBER, "
1520 BEEP
1530 GOTO Again
1540 !
1550 ! PUT THE NEW PRODUCT IN THE DATA BASE
1560 !
1570 Enter:DISP "€"
1580 INPUT "Enter product description", Desc$
1590 DBPUT (B$, "PRODUCT", 1, S(*), "@", Buf$)
1600 IF S(0) THEN Dberr
1610 DISP "NEW PRODUCT ADDED."
1620 ON KEY #1: "ANOTHER" GOTO Cont
1630 ON KEY #9 GOTO Cont
1640 WAIT
1650 Dberr:DISP LIN(2); "STATUS ERROR "; VAL$(S(0)); " IN LINE";
      S(6)
1660 GOTO Close
1670 Error:DISP LIN(2); "UNEXPECTED "; ERRM$
1680 GOTO Close
1690 Halt:DISP "*€ END OF ADD PRODUCT PROGRAM."
1700 Close:DBCLOSE (B$, "", 1, S(*))
1710 END

```

Example Program 5

Example program 5 allows products in the PRODUCT data set to be changed or deleted. The program prompts for the number of the product to be edited. The EXIT key may be pressed at any time to stop the program. The initial display is:

The initial display of the program is shown within a rounded rectangular frame. At the top center, the text "EDIT PRODUCT" is displayed. Below this, the prompt "Enter the number of the product you wish to edit." is shown, followed by a single hyphen "-" on the next line. At the bottom of the frame, there is a horizontal line above a row of eight rectangular input fields. The rightmost field in this row is labeled "EXIT PROGRAM".

If a number is entered for a non-existent product, it is detected by the calculated access DBGET in line 1330. If such a product number is entered, another number is requested:

The display after an invalid product number is entered is shown within a rounded rectangular frame. At the top center, the text "EDIT PRODUCT" is displayed. Below this, the prompt "Enter the number of the product you wish to edit." is shown. On the next line, the number "23" is entered. On the following line, the message "NO SUCH PRODUCT IN DATA BASE." is displayed. At the bottom of the frame, there is a horizontal line above a row of eight rectangular input fields. The rightmost field in this row is labeled "EXIT PROGRAM".

Note that the data base was opened in mode 1 (locking required before attempting to modify the data base). Line 1310 locks the data base. It is essential that the data base be locked before the DBGET. If it is locked afterwards, another user could lock the data base anytime before the lock in this program, and then make modifications to the record retrieved in line 1330. If such a user deleted the record, a subsequent DBUPDATE or DBDELETE would fail.

Once a correct product number has been provided, the old description is displayed for user edit:

```
EDIT PRODUCT

Enter the number of the product you wish to edit.
1

Enter New description
Uni-cycle_

ANOTHER  [ ]  [ ]  [ ]  [ ]  DELETE  [ ]  [ ]  EXIT
PROGRAM
```

The ANOTHER key may be pressed here to abort the modification and return to the initial display.

If the description is altered and **↑** is pressed, the DBUPDATE (line 1520) alters the text of the product description in the data base. In this case, either ANOTHER or EXIT PROGRAM is pressed to continue:

EDIT PRODUCT							
Enter the number of the product you wish to edit. 1							
Enter New description Unicycle UPDATE COMPLETE.							
ANOTHER							EXIT PROGRAM
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

If the DELETE key is pressed, the product is removed from the data base (see line 1670). The program then indicates that the delete was successful, and waits for the user to respond with the appropriate key:

EDIT PRODUCT							
Enter the number of the product you wish to edit. 1							
Enter New description Unicycle PRODUCT DELETED.							
ANOTHER							EXIT PROGRAM
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

If the entry in the PRODUCT master had any entries associated with it in the CUSTOMER detail, an error would have been issued (see line 1690).

Note that extreme care must be taken so that DBUNLOCKS are performed either after a successful operation or following an error (see lines 1350, 1550 and 1600). The DBCLOSE in line 1780 automatically performs the unlock in case either HALT or EXIT PROGRAM is pressed or an unforeseen error occurs.

```

1000 !   EXAMPLE PROGRAM 5
1010 !
1020 !   PRODUCT EDITOR
1030 !
1040   INTEGER S(9),Prod_no
1050   DIM B$(12),P$(10),Buf$(170)
1060   DIM Desc$(30)
1070   DISP "C";          ! CLEAR SCREEN
1080   DISP TAB(34);"EDIT PRODUCT"
1090   ON ERROR GOTO Error   ! SET UP ERROR AND HALT TRAPS
1100   ON HALT GOTO Halt
1110   B$="  SAD,SALES"
1120   P$="MANAGER"
1130   D$OPEN (B$,P$,1,S(*)) ! OPEN FOR SHARED ACCESS
1140   IF S(0) THEN Dberr
1150 !
1160 !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170 !
1180   DBASE IS B$
1190   IN DATA SET "PRODUCT" USE Prod_no,Desc$
1200 !
1210 !   INITIALIZE AND ASK FOR PRODUCT NUMBER
1220 !
1230   Not_found=17
1240   Chain_not_empty=44
1250   ON KEY #8:"EXIT" GOTO Halt
1260   ON KEY #16 GOTO Halt
1270 Cont:DISP " ";LIN(3);" ";
1280   OFF KEY #1,9
1290 Again:DISP " ";LIN(3);
1300   INPUT "Enter the number of the product you wish to edit.",
      Prod_no
1310   DBLOCK (B$,"",1,S(*)) ! LOCK DATA BASE BEFORE WRITE
1320   IF S(0) THEN Dberr
1330   DBGET (B$,"PRODUCT",7,S(*),"@",Buf$,Prod_no)
1340   IF S(0)<>Not_found THEN Maybe
1350   DBUNLOCK (B$,"",1,S(*))! UNLOCK DATA BASE AFTER AN ERROR
1360   IF S(0) THEN Dberr
1370   DISP "NO SUCH PRODUCT IN DATA BASE."
1380   BEEP
1390   GOTO Again
1400 Maybe:IF S(0) THEN Dberr
1410 !
1420 !   GET NEW DESCRIPTION AND PERFORM THE UPDATE
1430 !
1440   DISP "C"
1450   Desc$=TRIM$(Desc$)
1460   ON KEY #5:"DELETE" GOTO Del
1470   ON KEY #13 GOTO Del
1480   EDIT "Enter New description",Desc$
1490   OFF KEY #5,13
1500   DBUPDATE (B$,"PRODUCT",1,S(*),"@",Buf$)
1510   IF S(0) THEN Dberr
1520   DISP "UPDATE COMPLETE."
1530 Wait:DBUNLOCK (B$,"",1,S(*))! UNLOCK DATA BASE AFTER WRITE
1540   IF S(0) THEN Dberr
1550   ON KEY #1:"ANOTHER" GOTO Cont
1560   ON KEY #9 GOTO Cont
1570   WAIT

```

(continued)


```

1580 !
1590 !   DELETE THE ENTRY
1600 !
1610 Del:OFF KEY #5,13
1620   DBDELETE (B$, "PRODUCT",1,S(*))
1630   IF S(0)(<)Chain_not_empty THEN Del2
1640   DISP LIN(1);"THERE ARE STILL ORDERS FOR THIS PRODUCT."
1650   BEEP
1660   GOTO Wait
1670 Del2:IF S(0) THEN Dberr
1680   DISP LIN(1);"PRODUCT DELETED."
1690   GOTO Wait           ! GO DO UNLOCK.
1700 !
1710 !   TERMINATION ROUTINES
1720 !
1730 Dberr:DISP LIN(2);"STATUS ERROR ";VAL$(S(0));" IN LINE";
      S(6)
1740   GOTO Close
1750 Error:DISP LIN(2);"UNEXPECTED ";ERRM$
1760   GOTO Close
1770 Halt:DISP "*END OF EDIT PRODUCT PROGRAM."
1780 Close:DBCLOSE (B$,"",1,S(*))
1790   END

```

Utilities Examples

This section illustrates the two primary functions of the IMAGE/250 data base utilities. An example of data base backup and recovery is shown, as well as an example of data base restructuring.

Backup and Recovery

The best method of backup for many data bases involves duplicating all media on which the data base resides. The DUPL program on the SYSTEM disc is provided for this purpose. DUPL is described in Appendix C of the BASIC Programming Manual.

The DBSTORE and DBRESTORE utilities provide an alternative to disc duplication. The next figure shows how DBSTORE is used to backup the SAD data base.


```
LOAD BIN "DBSTOR"  
  
DBSTORE "SAD,TRANS" TO "SADBK" ON "BACK"  
* 2 5 6 1  
PLEASE INSERT VOLUME "SALES"  
3 4  
-
```



The SAD data base is stored on two discs, labeled TRANS and SALES. The TRANS volume contains the root file and data sets 2, 5, and 6. The SALES volume contains data sets 1, 3, and 4.

To backup this data base, the binary program is first LOADED from the SYSTEM disc. Then the volumes BACK and TRANS are inserted. Executing the DBSTORE command copies the root file and data sets 2, 5, and 6 from TRANS to the file SADBK on the volume BACK. DBSTORE then prompts the user to insert the SALES volume¹ so the remaining data sets can be stored. The TRANS volume will be replaced by the SALES volume. If the BACK volume had been replaced by SALES, instead of TRANS, the system would request the missing BACK volume:

```
LOAD BIN "DBSTOR"  
  
DBSTORE "SAD,TRANS" TO "SADBK" ON "BACK"  
* 2 5 6 1  
PLEASE INSERT VOLUME "SALES"  
PLEASE INSERT VOLUME "BACK"  
3 4
```

¹ After this volume has been inserted, press 

Once the data base has been backed up, normal operations involving it may be continued. Should the data base later become corrupt, it can be regressed to the state at the time of the last DBSTORE.

To restore a data base in its entirety, it is necessary to have a backup file containing ALL the sets in the data base. The SADBK backup file created previously contains all the data sets.

The first step in recovery is to purge the old copy of the data base using DBPURGE. The next figure shows how DBPURGE responds if all data sets are NOT available for the purge.

```
DBPURGE "SAD,TRANS"
2 5 6
( DATA SET "DATE", ON VOLUME "SALES" ) ERROR 77
( DATA SET "PRODUCT", ON VOLUME "SALES" ) ERROR 77
( DATA SET "LOCATION", ON VOLUME "SALES" ) ERROR 77
```

In this case only the volume TRANS was available. Since the SALES volume is not available, the data sets with numbers 2, 5, and 6 could not be purged. This means that the root file is still intact on the TRANS volume.

Inserting the SALES volume in addition to the TRANS volume and re-executing DBPURGE purges all remaining data sets and the root file, as shown below.

```
DBPURGE "SAD,TRANS"
( DATA SET "ORDER", ON VOLUME ":F2,6,0" ) ERROR 221
( DATA SET "OPTION", ON VOLUME ":F2,6,0" ) ERROR 221
( DATA SET "CUSTOMER", ON VOLUME ":F2,6,0" ) ERROR 221
1 3 4 *
```

The errors indicate that the named sets have already been purged. The * indicates that the root file has been purged.

To recover the backed up data base, the DBSTORE binary must be loaded from the SYSTEM disc. Then the TRANS volume is inserted, along with the BACK volume. When DBRESTORE is executed, all sets on the TRANS volume are first restored; then the user is prompted to insert the SALES disc. Once the SALES disc is inserted (replacing TRANS disc), the remaining sets are restored and the operation terminates.

The results of this restoration process are shown below.

```
LOADBIN DBSTOR, SYSTEM
DBRESTORE SADBK, BACK ON *, TRANS
* 2 5 6
PLEASE INSERT VOLUME SALES
1 3 4
```

-

Restructuring a Data Base

In order to restructure the data base, first unload it using the DBUNLD program on the SYSTEM disc. Once the data base has been unloaded onto a backup file, the data base itself is purged. Then the schema is altered (using EDITOR) and the root file regenerated via the Schema Processor (run SCHEMA). Once the data sets have been re-created (via DBCREATE), the DBLOAD program is used to reload the data stored in the backup file.

When the DBUNLD program is run, this form prompts for all information necessary to do the unload:

DATA BASE UNLOAD UTILITY
PARAMETER INPUT

Data Base Name Root File Volume Name

Maintenance Password

Unload from Data BASE Data Set Name

SERIAL Mode Unload

Backup File Name

List of Backup Volume Names

Please complete this form.

CHANGE SOURCE	CHANGE MODE		CLEAR FORM			ACCEPT INPUT	EXIT PROGRAM
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

The following is entered for the SAD data base:

DATA BASE UNLOAD UTILITY
PARAMETER INPUT

Data Base Name SAD Root File Volume Name TRANS

Maintenance Password

Unload from Data BASE Data Set Name

SERIAL Mode Unload

Backup File Name SADUNL

List of Backup Volume Names BACK

Please complete this form.

CHANGE SOURCE	CHANGE MODE		CLEAR FORM			ACCEPT INPUT	EXIT PROGRAM
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

The data base will be unloaded onto a file named SADUNL on the volume labeled BACK. Once the information has been entered and edited, if necessary, press the ACCEPT INPUT key to continue. The defaults concerning the source and mode are used, namely, unload entire data BASE and SERIAL mode unload.

The SYSTEM volume and TRANS volume should now be installed. DBUNLD opens the data base on TRANS. It then prompts for the BACK volume to create the backup file:

DATA BASE UNLOAD UTILITY UNLOAD PROCESSING							
Data Number	Data Number	Data Number	Data Number	Data Number	Data Number	Data Number	Data Number
Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded
Please insert volume BACK.							
							EXIT

The root file volume should be replaced with the BACK volume, since DBUNLD still needs information from the volume containing the DBUNLD program.

DBUNLD now prompts for the volume containing data set 3:¹

DATA BASE UNLOAD UTILITY UNLOAD PROCESSING							
Data Number	Data Number	Data Number	Data Number	Data Number	Data Number	Data Number	Data Number
Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded	Set Unloaded
Please insert volume SALES.							
							EXIT

¹ DBUNLD starts with data set 3 since data sets 1 and 2 are automatic master sets which are ignored for purposes of the unload.

SALES can replace any volume except the volume containing the backup file.

Once the first data set has been unloaded, the volume containing the DBUNLD programs must be re-inserted. Since there is no restriction concerning the volume name on which these programs reside, DBUNLD waits for the user to press before searching for the correct volume:

DATA BASE UNLOAD UTILITY
UNLOAD PROCESSING

Data Set	Number Unloaded	Data Set	Number Unloaded	Data Set	Number Unloaded	Data Set	Number Unloaded	Data Set	Number Unloaded
3	5								

Please insert volume SYSTEM.

							EXIT
<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>

The program alternately asks for the volume containing the next data set to unload and the volume containing the DBUNLD programs, until all data sets have been unloaded. The next screen is:

DATA BASE UNLOAD UTILITY
UNLOAD PROCESSING

Data Set	Number Unloaded	Data Set	Number Unloaded	Data Set	Number Unloaded	Data Set	Number Unloaded	Data Set	Number Unloaded
3	5								
4	10								
5	27								
6	11								

Please select a function.

						RESTART	EXIT PROGRAM
<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>

Press EXIT PROGRAM to terminate the unload routine.

Now the schema may be modified. Certain kinds of modifications from the following list may be made without a set-at-a-time reload:

- Change set capacity.
- Change length of a string item.
- Change sub-item count.
- Add items to or delete items from the end of a data set entry.
- Add sets to or delete sets from the end of the data base.¹

The next schema listing highlights changes made to the SAD data base:

- The length of the PROD-DESC field has been reduced to 24 characters, while the length of the REGION-DESC field has been increased to 40 characters.
- A new item called PROD-PRICE has been added; it occurs in the PRODUCT master and contains the base price of the product.
- The automatic master set DATE has been deleted. This change also requires that the path references to it on ORDER-DATE and SHIP-DATE in the CUSTOMER set be removed.
- The REGION-DESC and REGION-TYPE fields in the LOCATION set have been swapped to illustrate the item re-ordering option.
- The capacity of the OPTION set has been increased.

¹ When sets are deleted from the end of the data base, however, DBLOAD issues a warning about the loss of data.


```

$CONTROL      LIST, TABLE, ROOT
$TITLE "Sales Analysis Data Base"

BEGIN DATA BASE  SAD;  <<CUSTOMER SALES ANALYSIS DATA BASE>>

PASSWORDS:
      10      SALESMAN;
      15      MANAGER;
      3       SECRETARY; <<WILL HAVE READ ACCESS ONLY>>

ITEMS:
      ADDRESS,      2X30; <<2 LINES OF ADDRESS ALLOWED>>
      CITY,         X16;
      COUNTRY,      X12;
      DATE,         I; <<PATH FOR ORDER-DATE, SHIP-DATE>>
      NAME,         X30;
      OPTION-DESC,  X10;
      OPTION-PRICE, L;
      OPTION-TYPE,  I;
      ORDER-DATE,  I; <<MUST BE YYMM>>
      ORDER-NO,    X10;
      PRICE,       L;
      PRODUCT-NO,  I;
      PROD-DESC,   X24;
      REGION,      X6;
      REGION-DESC, X40;
      REGION-TYPE, I;
      SALESPERSON, X4;
      SHIP-DATE,   I; <<MUST BE YYMM>>
      STATE,       X6;
      ZIP-CODE,    X8;
      PROD-PRICE,  L;

SETS:

      NAME:      ORDER, A(3/10, 15);
      ENTRY:     ORDER-NO(2);
      CAPACITY:  101;

      NAME:      PRODUCT, MANUAL(3, 10/15), SALES;
      ENTRY:     PRODUCT-NO(1),
                PROD-DESC,
                PROD-PRICE;
      CAPACITY:  11;

      NAME:      LOCATION, M(3, 10/15), SALES;
      ENTRY:     REGION(1),
                REGION-TYPE,
                REGION-DESC;
      CAPACITY:  17;

```

```

NAME:    OPTION,D(3/10,15);
ENTRY:   ORDER-NO(ORDER),
         OPTION-DESC,
         OPTION-PRICE,
         OPTION-TYPE;
CAPACITY: 350;

```

```

NAME:    CUSTOMER,DETAIL(3/10,15);
ENTRY:   ORDER-NO(ORDER),
         NAME,
         ADDRESS,
         CITY,
         STATE,
         COUNTRY,
         ZIP-CODE,
         ORDER-DATE,
         SHIP-DATE,
         REGION(LOCATION),
         PRODUCT-NO(PRODUCT),
         PRICE,
         SALESPERSON;
CAPACITY: 100;

```

```
END.
```

```
UNREFERENCED ITEMS:
DATE
```

DATA SET NAME	TYPE	FLD CNT	PATH CNT	ENTRY LENGTH	MEDIA LENGTH	CAPACITY	SECTORS	VOLUME
ORDER	A	1	2	10	28	101	12	
PRODUCT	M	3	1	34	46	11	2	SALES
LOCATION	M	3	1	48	60	17	4	SALES
OPTION	D	4	1	30	34	350	47	
CUSTOMER	D	13	3	166	178	100	70	
							7	ROOT FILE LENGTH:
							142	TOTAL SECTORS INCLUDING ROOT:

```

NUMBER OF ERROR MESSAGES: 1
ITEM NAME COUNT: 21      DATA SET COUNT: 5
GLOBAL DBCB LENGTH: 1024 LOCAL DBCB LENGTH: 620

```

```
ROOT FILE SAD GENERATED.
```

Once these changes have been made to the schema text file, the Schema Processor is run to create the root file. Then DBCREATE is used to create the data set files. For the case of the restructured SAD data base, the result is:

```

DBCREATE 'SAD'; 'SECRET'
1 4 5 2 3
-

```

Once the new data has been created, the DBLOAD program can be run to reload the data base. DBLOAD initially prompts for the information required:

```

DATA BASE LOAD UTILITY
PARAMETER INPUT

Data Base Name ██████████      Root File Volume Name ██████████
Maintenance Password ██████████
Erase Data Base? (YES/NO) NO_
Load into Data BASE_          Data Set Name ██████████
Backup File Name ██████████    Backup File Set Number █
First Backup Volume Name ██████████
Re-order Items? (YES/NO) NO_

Please complete this form.

CHANGE ERASE  CHANGE DEST  CLEAR FORM  ACCEPT INPUT  EXIT PROGRAM
██████████  ██████████  ██████████  ██████████  ██████████

```

Since the restructuring requires that the sets be loaded one at a time, press the CHANGE DEST key (for load data set). The PRODUCT set is the first to be loaded. All information is entered into the form. DBLOAD then asks to verify the input. When everything is correct, YES is entered (or the ACCEPT INPUT key is pressed).

```

DATA BASE LOAD UTILITY
PARAMETER INPUT

Data Base Name SAD_          Root File Volume Name TRANS_
Maintenance Password SECRET_
Erase Data Base? (YES/NO) NO_
Load into Data SET_        Data Set Name PRODUCT_
Backup File Name SADUNL_    Backup File Set Number 3_
First Backup Volume Name BAC_
Re-order Items? (YES/NO) NO_

Is all information correct? (YES/NO)
YES_

EXIT
██████████  ██████████  ██████████  ██████████  ██████████  ██████████  ██████████

```

DBLOAD then prompts for the volume containing the backup file, in this case, the volume named BACK:

DATA BASE LOAD UTILITY LOAD PROCESSING													
Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
Please insert volume BACK.													EXIT

The user is then asked to enter the data set volume:

DATA BASE LOAD UTILITY LOAD PROCESSING													
Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
Please insert volume SALES.													EXIT

The data from the backup file is then put into the data set, and the user is prompted for the root file volume, so the root file can be updated to reflect the contents of the data set:

DATA BASE LOAD UTILITY
LOAD PROCESSING

Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
2	5								

Please insert volume TRANS.

Finally, the user is prompted for the volume containing the DBLOAD program:

DATA BASE LOAD UTILITY
LOAD PROCESSING

Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
2	5								

Please insert volume SYSTEM.

Loading the PRODUCT data set is now complete.

To load the next data set, the RESTART key is pressed to get back into the parameter input phase. The next data set is called LOCATION, data set number 4 on the old version of the data base (it is now data set number 3). Also, since the order of some of the items has changed, the "re-order items" option must be selected. After entering this data, the screen is:

DATA BASE LOAD UTILITY
PARAMETER INPUT

Data Base Name **RAD** Root File Volume Name **TRANS**

Maintenance Password **SECRET**

Erase Data Base? (YES/NO) **NO**

Load into Data **SET** Data Set Name **LOCATION**

Backup File Name **SADJNL** Backup File Set Number **4**

First Backup Volume Name **BACK**

Re-order Items? (YES/NO) **YES**

Please complete this form.

CHANGE ERASE	CHANGE DEST	CHANGE RE-ORDER	CLEAR FORM			ACCEPT INPUT	EXIT PROGRAM
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

When the ACCEPT INPUT key is pressed, the user is asked to enter the new item order. Since items two and three were swapped, the values entered are:

DATA BASE LOAD UTILITY
PARAMETER INPUT

New	Item	Order										
1	1	14	27	40	53	66	79	92	105	118		
2	3	15	28	41	54	67	80	93	106	119		
3	2	16	29	42	55	68	81	94	107	120		
4		17	30	43	56	69	82	95	108	121		
5		18	31	44	57	70	83	96	109	122		
6		19	32	45	58	71	84	97	110	123		
7		20	33	46	59	72	85	98	111	124		
8		21	34	47	60	73	86	99	112	125		
9		22	35	48	61	74	87	100	113	126		
10		23	36	49	62	75	88	101	114	127		
11		24	37	50	63	76	89	102	115			
12		25	38	51	64	77	90	103	116			
13		26	39	52	65	78	91	104	117			

Please enter new item order (data set item position, **backup file item position**).

		FILL FORM	CLEAR FORM			ACCEPT INPUT	EXIT
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

The ACCEPT DATA key is then pressed, and volumes are requested as done previously for data set PRODUCT.

When loading the LOCATION set is complete, the screen appears as below and RESTART is pressed to load the next set.

DATA BASE LOAD UTILITY LOAD PROCESSING											
Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded		
3	10										
Please select a function.											
								RESTART	EXIT PROGRAM		

The next set is the OPTION set. The "Re-order items" option should be turned off. The responses in the parameter input section are:

DATA BASE LOAD UTILITY PARAMETER INPUT							
Data Base Name	SAD	Foot File Volume Name	TRANS				
Maintenance Password	SECRET						
Erase Data Base? (YES/NO)	NO						
Load into Data	SET	Data Set Name	OPTION				
		Backup File Set Number	5				
Backup File Name	SADJHL	First Backup Volume Name	BAC+				
Re-order Items? (YES/NO)	NO						
Please complete this form.							
CHANGE ERASE	CHANGE DEST	CHANGE RE-ORDER	CLEAR FORM			ACCEPT INPUT	EXIT PROGRAM

When ACCEPT INPUT is pressed, the DBLOAD program requests various volumes and loads the data set. The next screen is:

DATA BASE LOAD UTILITY LOAD PROCESSING							
Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
4	27						
Please select a function.							
						RESTART	EXIT PROGRAM

Pressing RESTART returns to the PARAMETER INPUT section which, for the CUSTOMER set, is entered as shown.

DATA BASE LOAD UTILITY PARAMETER INPUT							
Data Base Name	SAD	Pool File Volume Name	TPR05				
Maintenance Password	SECRET						
Erase Data Base? (YES/NO)	NO						
Load into Data	SET	Data Set Name	CUSTOMER				
		Backup File Set Number	8				
Backup File Name	SAD008	First Backup Volume Name	B001				
Re-order Items? (YES/NO)	NO						
Please complete this form.							
CHANGE ERASE	CHANGE DEST	CHANGE RE-ORDER	CLEAR FORM			ACCEPT INPUT	EXIT PROGRAM

When ACCEPT INPUT is pressed, the system begins requesting volumes to allow the load. However, since a DBPUT into the CUSTOMER set requires both the PRODUCT master and OPTION master to be simultaneously available, both the SALES and TRANS volumes must be installed for a successful DBPUT. On a system with only two disc drives, this means that the first entry in the backup file must be read, then the backup volume must be removed and the proper data base volumes inserted to allow the DBPUT. Then another entry must be obtained from the backup file. To do this, DBLOAD requires certain routines from the volume containing the DBLOAD programs.

Thus, for each entry in the OPTION set, many volume exchanges are required. This process can be greatly simplified by any of the following means:

- Put the entire data base all on the same volume.
- Put the backup file on same volume as either the DBLOAD program or one of the data base volumes.
- Put DBLOAD program on one of the data base volumes.

After the CUSTOMER data set has been loaded, the CRT appears as shown below. Press EXIT PROGRAM to terminate the load procedure.

DATA BASE LOAD UTILITY
LOAD PROCESSING

Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded	Data Set	Number Loaded
5	11								

Please select a function.

								RESTART	EXIT PROGRAM

The SAD data base is now restructured.



Data Base Locking

Locking is a means of communication and control used by mutually cooperating programs. A lock on a particular section of the data base prevents other programs from modifying that section. DBLOCK is used only on data bases opened in mode 1, shared write access. In access modes 3 and 8, DBLOCK is ignored.

As implemented in the IMAGE2 DROM, the DBLOCK statement operates in one of twelve modes. Modes 1 through 6 apply a write lock to the section specified; modes 11 through 16 apply a read lock to the section specified. Modes 1, 2, 11 and 12 are used to lock an entire data base. Modes 3, 4, 13 and 14 are used to lock a data set. Modes 5, 6, 15 and 16 are used to lock an entry or group of entries specified by a lock descriptor.

Before adding, deleting or modifying a data entry, a program must acquire a write lock covering that entry. The locks required are:

- To modify (DBPUT, DBDELETE, DBUPDATE) a data entry in a detail data set, a current lock must cover the affected data entry. It may be a data entry, data set or data base level lock.
- To add to (DBPUT) or delete from (DBDELETE) a master data set, a current lock must cover the data set or data base. To update (DBUPDATE) a master data set, data entry level locks are sufficient.

A program can read a section of the data base without locking it even if that section is locked by another program. If a second program is modifying the data base during the read, unexpected results can occur. For example, while a program is performing chained GET's, the address of the next entry can be modified by a second program. To prevent this, the appropriate data set or entries in the data set should be locked.

Locks used to protect read operations should be read locks. A read lock prevents other programs from getting a write lock on an entry, thus preventing any modification of that entry. A read lock will not prevent other programs from getting a read lock on the same entry. The shared read modes allow for greater access to the data base while preventing write operations.

The DBLOCK operation makes no modifications to the data base itself. The entries locked do not have to exist in the data base. This will be the case when a new entry is created by a DBPUT.

Lock Descriptors

In modes 5, 6, 15 and 16, the program specifies the entries to be locked through a lock descriptor. A descriptor consists of a set name, a relational operator and an associated value. (Refer to DBLOCK in Chapter 3 for a complete description). Multiple descriptors can be concatenated to build complex locks. The string of lock descriptors, call a **lock predicate**, is passed to the locking system in the QUALIFIER parameter of DBLOCK.

If @ is specified for the data set name, IMAGE/250 interprets this to mean “lock all data sets” (i.e., the whole data base. This is equivalent to the operation of modes 1, 2, 11 and 12). Similarly, if @ is specified for the item name, the interpretation is “lock all items” in the specified data set (i.e., lock that data set; equivalent to modes 3, 4, 13 and 14).

IMAGE/250 also allows set name to be @ with item name, relational operator and value specified for a desired entry value. This means “lock this entry wherever it occurs in any set”.

A lock descriptor is a string expression with a very specific format:

Length	One word integer containing the physical length in words of this descriptor. The length includes the length parameter itself.
Set Name	Eight words containing the name of an existing IMAGE/250 data set up to 15 bytes long padded on the right with blanks, or a set number stored as a binary integer from 1 thru 50 in the first word; the remaining seven words are ignored. If the first (leftmost) byte of the name is @, then the remaining 15 bytes are ignored and IMAGE/250 interprets this to mean “all sets in the data base”. If the first word of this field is a binary zero, the whole predicate is ignored.
Item Name	Eight words containing the name of an existing IMAGE/250 data item (need not be a key item) up to 15 bytes long padded on the right with blanks or an item number stored as a binary integer from 1 thru 255 in the first word; the remaining seven words are ignored. If the first (leftmost) byte of this field is @, then the remaining 15 bytes are ignored. IMAGE/250 interprets the @ to mean “all items in the set” (i.e., the whole set); the value field is ignored.

Relop One word containing one of these relational operators stored in its ASCII representation:

= equal	} ASCII Comparison
>= greater than or equal	
<= less than or equal	

EQ equal	} Lexical Comparison
GE greater than or equal	
LE less than or equal	

For the equals operator, the operator character may appear in either byte and the other byte must be a blank character (octal 40).

Value The value of the specified item to be locked. It should be stored exactly as stored within the data base. However, for numeric types, IMAGE/250 will perform any necessary conversion, if possible. If a conversion error is caused by a length incompatibility or size incompatibility, IMAGE/250 returns a status error. IMAGE/250 uses as many words as required by the corresponding item definition. If a string value is given that is shorter than that specified for the item, blanks are added as needed. If the string value is longer, it is truncated.

The different relational operators for lexical comparison and ASCII comparison are provided for systems supporting foreign languages. On the HP 250, the relational operators =, >= and <= compare strings based on their ASCII collating sequence. On foreign language systems, it is more often the lexical comparison that is of interest. For example, in German the strings "mueler" and "mÜler" are equal. In ASCII, however, the relation "mueler" < "mÜler" is true. Similarly, in Spanish the string "chico" is greater than the string "color" in the local language order. This obviously would not be true in ASCII.

The HP 250 supports string comparison in foreign languages through the EUROPE DROM. To use this capability in the Record Locking System, the relational operators EQ, GE and LE must be specified in the predicate description. In systems that do not have the EUROPE DROM, these relational operators will default to their equivalent operators based on ASCII comparison.

Evaluating conflicts between descriptors using lexical relational operators and those using ASCII relational operators causes special problems. Since the relationships between collating sequences can be very complex, the IMAGE/250 Record Locking System will not allow descriptors with both types of relational operators to be in effect at the same time. This restriction applies only to descriptors with string values because the two classes of relational operators are equivalent for numeric values.

Setting-up Predicates

The predicates required by DBLOCK in modes 5, 6, 15 and 16 may be created by several methods. Simply building a string expression with concatenation may be the easiest approach. For example:

```
CHR$(0) & CHR$(21) & Set$ & Item$ & "<=" & "GEORGE"
```

The one-word integer-length field is formed by two-byte values using the CHR\$ function. The length value must be carefully specified to include the complete descriptor. The Set\$ and Item\$ strings must have a current length of 16.

The PACK statement can also be used to construct a predicate. The next sequence could be used in a program to lock PRODUCT-NO=16117 in the data set PRODUCT.

```
100  INTEGER N,Product_no,Stat(9)
110  DIM Q$(40),Lock_set$(16),Lock_item$(16),Relop$(2)
120  Product_no=16117
130  Lock_item$="PRODUCT-NO      "
140  Lock_set$="PRODUCT          "
150  Relop$="="
160  N=19
170  Mode=5
180  PACK USING 190;Q$
190  PACKFMT N,Lock_set$,Lock_item$,Relop$,Product_no
200  DBLOCK (Base$,Q$,Mode,Stat(*))
   :
```

The disadvantage of these methods is that the programmer must manually specify the descriptor length and guarantee the correct length for each field. The PREDICATE statement computes the descriptor length and insures that the format is correct. To create the descriptor in the previous example, the programmer need only write:

```
500 PREDICATE Q$ FROM "PRODUCT", "PRODUCT-NO", "=", 16177
```

The programmer can often choose from many equivalent lock sequences. For example, to apply a read lock to the data set LOCATION, the programmer could use any of the following sequences:

```
200 INTEGER N, Stat(9)
210 DIM Lock_set$[16], Lock_item$[16], Q$[40]
220 Lock_item$="@ "
230 Lock_set$="LOCATION"
240 N=17
250 PACK USING 170; Q$
260 PACKFMT N, Lock_set$, Lock_item$
270 DBLOCK (Base$, Q$, 15, Stat(*))
:
:
or
200 Mode=13
210 Lock_set$="ITEM-MASTER"
220 DBLOCK (Base$, Lock_set$, Mode, Stat(*))
:
:
or
200 PREDICATE P$ FROM "ITEM-MASTER", "@"
210 DBLOCK (Base$, P$, 16, Stat(*))
:
:
```

Multiple descriptors can be combined in a predicate to specify complex locks. Descriptors can be combined by concatenation or by specifying multiple set-item-value groups on the PREDICATE statement. The following example locks the data set LOCATION as well as all values of PRODUCT-NO less than or equal to 10,000 in the data set PRODUCT.

```

110  INTEGER Lock_itemnum,Stat(9)
120  DIM Q$(40),Lock_set$(1:2)[16],Relop$(2)
130  Lock_item$="@ "
140  Lock_itemnum=14          ! Item #14 for PRODUCT-NO.
150  Lock_set$(1)="LOCATION   "
160  Lock_set$(2)="PRODUCT  "
170  Relop$="<="
180  Keyinfo=10000
190  Mode=5
200  PREDICATE Q$ FROM Lock_set(1),Lock_item$;Lockset$(2),
Lock_itemnum,Relop$,Keyinfo
210  DBLOCK (Base$,Q$,Mode,Stat(*))
:
:

```

The maximum length of a predicate string is 4095 bytes.

Lock Conflicts

The locking system recognizes and acts upon the relationships that implicitly exist between lock descriptors. For example, an attempt to lock a data set must wait if the whole data base is locked or if an item/value within the set is locked.

It is also necessary to restrict entry lock requests such that at any one time only a single item name in a data set may be used for locking purposes. Lock requests on different values of the same item are acceptable. However, the locking system cannot determine if the collection of entries locked using different item names have any entries in common. In this case, DBLOCK assumes that a conflict exists and queues the later request or returns a status error.

When a request is queued, no other request which would conflict with the request in queue is granted. For example, assume program A has locked a data entry in data set X, and program B wants to lock data set X and data set Y. Program B's request is queued. When program C requests to lock a data entry in data set Y, that request is queued because program B is waiting to lock data set Y.

When using DBLOCK wait modes, the programmer should be careful to avoid possible deadlock conditions. If a program makes multiple resource requests using the commands DBLOCK, LOCK# or REQUEST, a potential deadlock may occur if another program is also making requests for the same resources. For example, suppose program A holds resource 1 and is queued waiting for resource 2 which is held by program B. If program B makes a wait request for resource 1, a deadlock situation occurs. If a request for a resource would cause a deadlock, the operating system notifies the requesting program with error 1002. Programs requesting the same resources can avoid deadlock by making their resource requests in the same order.

The following table summarizes the conditions for granting a lock.

Lock Request	Conflicting Lock	Action
Whole data base write-lock	● whole data base already write-locked.	Wait
	● whole data base already read-locked.	Wait
	● one or more sets write-locked.	Block* and wait
	● one or more sets read-locked.	Block* and wait
	● one or more item / values write-locked.	Block* and wait
	● one or more item / values read-locked.	Block* and wait
Whole data base read-lock	● whole data base already write-locked.	Wait
	● whole data base already read-locked.	Grant lock
	● one or more sets write-locked.	Block* and wait
	● one or more sets read-locked.	Grant lock
	● one or more item / values write-locked.	Block* and wait
	● one or more item / values read-locked.	Grant lock
Whole data set write-lock	● whole data base already write-locked.	Wait
	● whole data base already read-locked.	Wait
	● requested set write-locked.	Wait
	● requested set read-locked.	Wait
	● one or more item / values in requested set write-locked.	Block* and wait
	● one or more item / values in requested set read-locked.	Block* and wait
Whole data set read-lock	● whole data base already write-locked.	Wait
	● whole data base already read-locked.	Grant lock
	● requested set write-locked.	Wait
	● requested set read-locked.	Grant lock
	● one or more item / values in requested set write-locked.	Block* and wait
	● one or more item / values in requested set read-locked.	Grant lock

Cont.

Blocked means that no more locks capable of impeding the request will be granted.

Lock Request	Conflicting Lock	Action
Item / value in set write-lock	● whole data base already write locked.	Wait
	● whole data base already read-locked.	Wait
	● set write-locked.	Wait
	● set read-locked.	Wait
	● requested item / value write-locked.	Wait
	● requested item / value read-locked.	Wait
	● different item in set write-locked.	Block* and wait
	● different item in set read-locked.	Block* and wait
Item / value in set read-lock	● whole data base already write locked.	Wait
	● whole data base already read-locked.	Grant lock
	● set write-locked.	Wait
	● set read-locked.	Grant lock
	● requested item / value write-locked.	Wait
	● requested item / value read-locked.	Grant lock
	● different item in set write-locked.	Block* and wait
	● different item in set read-locked.	Grant lock

*"Blocked" means that no more locks capable of impeding the request will be granted.

Defaults Without IMAGE2

The locking system described in this section requires that IMAGE2 DROM is configured. However, a program using DBLOCK may be written so that it operates with or without the IMAGE2 DROM. This could be useful in programs used on both single-user and multi-user systems.

Without the IMAGE2 DROM, DBLOCK defaults all lock requests to whole data base locks. That is, all even modes default to mode 2 and all odd modes default to mode 1. Also, status errors resulting from conflicts default to 20.

Since the PREDICATE statement is in the IMAGE2 DROM, programs using PREDICATE require IMAGE2. If a program is to run without IMAGE2, other means of setting up predicates must be used.

APPENDIX A

PACK Statements

Introduction

The **PACK DROM** contains three statements which provide a convenient means of transferring string and numeric data to and from a string variable. **UNPACK** is particularly useful in conjunction with certain **DBINFO** modes, in which data base information is returned in a string variable as a combination of ASCII characters and numeric integers.

The **PACK** statements are:

<code>PACKFMT</code>	Specify the data format for PACK and UNPACK .
<code>PACK</code>	Transfer data from variables in a pack list to a destination string.
<code>UNPACK</code>	Transfer data from a source string to variables in a pack list.

The **PACKFMT** Statement

`PACKFMT` pack list

The parameter is:

pack list	A list of program variables, arrays and/or skip fields separated by commas. This list contains an ordered set of variable names used by the PACK and UNPACK statements.
-----------	---

PACKFMT (pack format) defines a list of variables to be used in conjunction with a source or destination string referenced in an **UNPACK** or a **PACK** statement. Upon **PACK** execution, data is transferred from the **PACK** list variables to the destination string. Upon **UNPACK** execution, data is transferred from the source string to the pack list variables.

As the transfer occurs between the pack list variables and the string referenced in **PACK** or **UNPACK**, an internal pointer to the string's next position is updated. To skip character positions within the string, a skip indicator may be supplied in the appropriate position of the pack list (e.g., `1X` = one byte, `2X` = two bytes). **PACK** and **UNPACK** program examples illustrate the use of the **PACKFMT** statement and the skip indicator.

The PACK Statement

PACK USING line id destination string

The parameters are:

- destination string A string variable that receives data contained in variables listed in a PACKFMT statement.
- line id A line number or line label referencing the pack list.

The PACK statement transfers data from each variable of the appropriate pack list to the destination string. The pack list is located on a separate program line. As the data is transferred to the destination string, its format is not altered in any way. Thus, a real-precision number requires eight bytes in the buffer; a short-precision number requires four bytes; etc. When transferring a string from the pack list having a current length less than its dimensioned (or substring) length, the destination string is filled with blanks to equal the dimensioned (or substring) length.

The following example illustrates the use of the PACK statement.

```
10  INTEGER A
20  SHORT B
30  REAL C
40  DIM D$(10),E$(50)
50  A=47
60  B=89.5432
70  C=2.3456789
80  D$="IMAGE "
90  PACK USING Here,E$
100 END
110 Here:  PACKFMT A,B,2X,D$,14X,C
```

After executing line 90, E\$ contains the following:

Bytes	Value
1-2	integer variable A
3-6	short variable B
7-8	skipped
9-18	string variable D\$ (last 5 bytes are padded with spaces)
19-32	skipped
33-40	real variable C
41-50	filled with spaces

The UNPACK Statement

UNPACK USING line id # source string

The parameters are:

- source string A string expression that contains data to be unpacked into variables listed in a PACKFMT statement.
- line id A line number or line label referencing a PACKFMT statement.

Through the UNPACK statement, data is transferred from the source string to the variables appearing in the pack list. A one-to-one transfer is done without altering the data format. When transferring data to short- or real-precision variables, UNPACK verifies that the data qualifies as a valid numeric value.

Here's an example use of UNPACK:

```
10  DIM A$[12],F#[40]
20  INTEGER X1,X2
   .
200 UNPACK USING 230;F#
210 DISP X1,A#,X2
220 END
230 PACKFMT 4X,X1,A#,X2
```

After executing line 200, variables X1, X2 and A\$ contain the following information:

Variable	Bytes in F\$
X1	5-6
X2	19-20
A\$	7-18

APPENDIX B

HP 250 Text Editor

Introduction

The HP 250 EDITOR program is used to create and maintain data files containing lines of text. The primary purpose of the EDITOR is to build and modify data base definitions (schemas). EDITOR may also be used to edit files containing only string data, such as files produced by the SAVE statement.

The EDITOR program does not make changes to existing data files directly. Instead, a copy of the file is maintained in memory and in two scratch files. This copy of the file is known as the **work file**. All additions, modifications, and deletions are made only to the work file. The work file may be copied to a new or existing data file at any time.

EDITOR organizes the work file into pages (blocks). A page can contain from 5 thru 200 lines, depending on the available user memory size. Pages are automatically loaded into memory and copied to the scratch files as needed. Lines within the page in memory may be accessed quickly, while all other lines must be located on the scratch files and loaded into memory. Thus, editing time may be significantly reduced by making changes to lines in ascending line-number order.

To run the EDITOR program, execute the command:

```
RUN "EDITOR [volume spec]"
```

The volume spec parameter must appear when the EDITOR program is not on the default mass storage device. Following the RUN command, the EDITOR displays:

```
HP250.1.A          TEXT EDITOR
  ~~~~~
  software revision level
```

The slash (/) following the heading indicates the EDITOR is ready to execute commands. Commands may be entered in upper or lower case. Several commands may be entered on one line by separating commands with semicolons (;). The total command line cannot exceed 160 characters (two display lines). EDITOR commands are described later in this appendix.

During normal operations, two scratch files, \$ED\$xA and \$ED\$xB, are used to store portions of the work file. A third file, \$ED\$xH, is created if a HOLD command is executed.¹ These files are created on the default mass storage device, which must remain on-line while the program is running. Each file requires a minimum of 80 sectors, and may require more disc space when editing large data files.

All data in the work file is stored and retrieved as lines. Each line is assigned a unique line number from .001 thru 9999.999.

Lines are normally 80 characters or less, but may be as long as 160 characters² (two display lines). When the line number is displayed along with the line, the number is displayed in half bright to distinguish it from the line.

EDITOR commands operate on a single line or on groups of lines. Individual lines are specified by a single line number. In addition, the first and last lines of the work file can be specified by using the words FIRST or LAST instead of a line number. A group (range) of lines is specified by two line numbers separated by a slash (/). All lines in a work file are specified by the word ALL. Some examples of line and range specifiers are:

1	Specifies line one.
FIRST	Specifies the first line in the work file.
1/10.5	Specifies all lines from 1 thru 10.5.
FIRST/LAST	Specifies all lines in the work file.
ALL	Specifies all lines in the work file.

Error Messages

Two different kinds of error messages are reported by EDITOR. Normal errors are reported when the given command cannot be performed (e.g., SYNTAX ERROR). Warning messages are displayed when special conditions are encountered (e.g., LINE TRUNCATED), but do not interfere with the execution of the command. Error messages are generated using a special error message file, EDERRS, which is on the SYSTEM disc. If the SYSTEM disc containing the EDERRS file is not on-line, error messages will have the form:

--- ERROR --- error number

A list of error numbers and their meaning is in Appendix E.

¹ In each file name, the letter x will be replaced by a number from 1 thru 6.

² The Schema Processor reads and prints only the first 80 characters of a line, and processes only the first 72 characters of a line, regardless of the actual line length.

Special Control Keys




is used to execute all EDITOR commands.






clears the line just typed and positions the cursor at the left margin.



terminates an edit operation.  can be used to terminate the ADD, CHANGE, DELETE, FIND, HOLD, LIST and MODIFY commands.



used to examine the value of certain EDITOR parameters. The total number of lines in the work file are displayed by typing `Lines` . Type `Length`  to display the maximum number of characters per line. Type `LP`  to display the number of lines per page output on an offline listing.

EDITOR Commands

The following commands are used with EDITOR to edit the work file. The same conventions used earlier to describe IMAGE/250 syntax are used here.

ADD	Adds lines to the work file.
CHANGE	Changes character strings in the work file.
DELETE	Deletes lines from the work file.
END	Terminates the EDITOR program.
FIND	Finds specified character strings or current line position.
GATHER	Renumbers a work file.
HOLD	Saves lines from the work file into the hold file.
KEEP	Saves the work file as a data file.
LIST	Lists lines from the work file to the display or printer.
MODIFY	Modifies lines in the work file.
SET	Sets EDITOR parameters.
TEXT	Copies a data file into the work file.
WHILE	Repeats a group of EDITOR commands.

The ADD Command

```
{ A }  
{ ADD } [Q][line number][, HOLD]
```

The ADD command adds lines of text into the work file. Lines may be entered from the keyboard or from the HOLD file. Entering two slashes (//) or pressing **HALT** terminates the ADD command.

If no options are specified, the line number of the line to be added is displayed (in half bright), and the cursor is positioned after the number in preparation for input from the keyboard. Lines are added directly after any existing lines in the work file. Subsequent lines are numbered in increments of 1. If the Q (quiet) parameter is specified, no line numbers are displayed.

If a line number is specified, lines are added starting at the specified line. Subsequent lines are added in increments of 1, .1, .01 or .001, depending on the line number specified and the next higher line in the work file. The specified line number must be numeric, and not reference an existing line number. The line number parameter allows lines to be added anywhere in the work file.

Specifying HOLD allows lines of text to be added from the hold file into the work file. Lines from the hold file are numbered as if they were entered from the keyboard.

Two examples of this command are:

```
ADD 5.1
```

Adds line 5.1 into the work file. Subsequent lines are numbered in increments of .1, .01, or .001, depending on the number of the next line in the work file.

```
ADD, HOLD
```

Adds lines of text from the hold file. Lines are inserted at the end of the work file. All lines in the hold file are added unless either **HALT** is pressed or an error occurs.

The CHANGE Command

```
{ CHANGE }  
  C      } [Q] string1 TO string2 [ IN range list]
```

The CHANGE command replaces character strings within specified lines. Both string₁ and string₂ may be any ASCII string, and must be delimited by any non-alphanumeric character¹ not appearing in the string.

If no options are specified, all occurrences of string₁ are replaced with string₂ in the current line. The line is then displayed if any replacements were made. If Q (quiet) is specified, the line is not displayed.

If a range is specified, all lines within the specified range that contain string₁ are changed. Changed lines are displayed if Q is not specified. The change operation is terminated by pressing **HALT**.

Some examples of this command are:

```
C "ABC" TO "CBA" IN 1/5,8,9/13
```

Changes all occurrences of ABC to CBA in lines 1 thru 5, line 8, and lines 9 thru 13. All changed lines are displayed.

```
CHANGE Q "ABC" TO "DEF" IN 1
```

Changes all occurrences of the string ABC to DEF in line 1. Line 1 is not displayed.

```
CHANGE "ABC" TO "ABC" IN ALL
```

Displays all lines containing the string ABC.



¹ The string delimiter must be a single character, and cannot be a space, semicolon (;), alphabetic character (A thru Z, a thru z) or a number (0 thru 9).

The DELETE Command

```
{ DELETE } [Q][range list]
  D
```

The DELETE command deletes lines from the work file. Deleted lines are not recoverable.

If no parameters are specified, the current line is displayed and deleted. If Q (quiet) is specified, the line is not displayed.

If a range is specified, all lines within the specified range are deleted. Deleted lines are displayed if Q is not specified. The delete operation may be terminated by pressing **HALT**.

Some examples of this command are:

```
DELETE 5
```

Displays and deletes line 5 from the work file.

```
DELETE 5/LAST
```

Displays and deletes all lines from line 5 to the last line in the work file.

```
DQ 5/7,9/13,15
```

Deletes lines 5 thru 7, lines 9 thru 13, and line 15. No lines are displayed.

The END or EXIT Command

```
{ E  
EXIT  
END }
```

The END or EXIT command terminates the EDITOR program and returns control to the operating system. All scratch files used to store the work file are purged. If any modifications have been made to the work file without executing a KEEP command, the EDITOR requests confirmation before purging the work file.

For example, entering EXIT terminates the edit session. If any modifications have been made to the work file, the program displays:

```
If it is okay to clear type "YES".  
Clear?
```

If a Y or YES is entered, the program clears the work file, purges the scratch files, and displays:

```
END OF EDITOR PROGRAM.
```

The FIND Command

$$\left\{ \begin{array}{l} F \\ \text{FIND} \end{array} \right\} [\text{Q}] \left[\begin{array}{l} \text{string IN range list} \\ \text{line number} \end{array} \right]$$

The first form of the FIND command is used to locate a specified character string in the work file and to position the current line pointer to that line. If no options are specified, the work file is scanned for the first occurrence of the specified character string, starting with the current line. Lines preceding the current line are not searched. The character string may be any ASCII character string, delimited by any non-alphanumeric character¹ not appearing in the string.

If a range list is specified, the line or lines specified are scanned for the first occurrence of the character string. If the character string is found, that line is displayed and the line pointer is set to that line. If the character string is not found, a message is displayed, and the line pointer is set to the line following the last line scanned. If Q is specified, the line containing the character string is not displayed.

When only a line number is specified (in the second form of FIND), the line pointer is set to the specified line, and the line is then listed. If Q is specified, the pointer is set without displaying the line. If a line number is not specified, the current line is listed without advancing the line pointer.

Some examples of this command are:

```
FIND"ABC" IN 5/15
```

Lists the first line in the given range (lines 5 thru 15) that contains the string ABC and sets the line pointer to that line.

```
FIND FIRST
```

Resets the line pointer to the first line in the work file, and displays that line.

```
F"XYZ" IN ALL
```

Displays the first line in the work file that contains the string XYZ. The current line pointer is set to the line displayed.

¹ The string delimiter must be a single character, but cannot be a space, semicolon (;), alphabetic character (A thru Z, a thru z) or a number (0 thru 9).

The GATHER Command

```
{ G  
GATHER } ALL [TO line number [BY increment value] ]
```

The GATHER command renumbers the entire work file. If the line number and increment value are not specified, lines are numbered in increments of 1 starting with the value 1.

If a line number is specified, the first line is renumbered with the value specified. If an increment value is specified, it is used as the incremental value for the renumbering process instead of the default value of 1.

An example of this command is:

```
GATHER ALL TO 100 BY 10
```

Renumbers the work file in increments of 10. The first line number in the work file is assigned line number 100.

The HOLD Command

```
{ H }  
HOLD [Q][range][, APPEND]
```

The HOLD command copies lines from the work file to the hold file. Lines saved in the hold file may be added into the work file using the ADD command. Groups of lines may be moved within the work file using the HOLD, DELETE and ADD commands.

If no parameters are specified, the hold file is cleared, and the current line is copied into the hold file. If the Q (quiet) option is not specified, the copied line is also displayed.

If a range is specified, all lines within the specified range are copied into the hold file. Copied lines are displayed unless Q is specified. The hold operation is terminated by pressing .

If APPEND is not specified, the hold file is cleared before copying lines. If the hold file contains any lines, EDITOR requests confirmation before clearing the hold file. If APPEND is specified, the specified lines are appended to the end of the hold file.

Some examples of this command are:

```
HOLD 5/10, APPEND
```

Copies lines 5 thru 10 to the end of the hold file. Existing lines in the hold file are unaffected.

```
HOLD 5/10; DELETE 5/10; ADD; HOLD
```

Moves lines 5 thru 10 to the end of the work file. Before clearing the hold file, the program displays CLEAR HOLD?. A response other than Y or YES terminates the command without affecting the contents of the hold or work files.

The KEEP Command

$$\left\{ \begin{array}{c} K \\ \text{KEEP} \end{array} \right\} \text{ file spec } \left[\begin{array}{l} ; \text{UNN} \\ ; \text{UNNUMBERED} \end{array} \right]$$

The **KEEP** command saves the contents of the work file in a file specified by the file spec. The file specifier must be enclosed in quotes. If the file already exists, the old file is purged before the file is kept. The **EDITOR** requires confirmation from the user before the old file is purged. If the old file is protected (files can be protected using the **BASIC** command **PROTECT**), the correct protect code must be entered before **EDITOR** can purge the old file.

When **UNN** or **UNNUMBERED** is specified, lines are saved without line numbers. If this option is not specified, blanks are appended to the end of each line to fill the maximum number of characters per line, followed by an 8-character line number. The Schema Processor accepts either numbered or unnumbered files.

Two examples of this command are:

```
KEEP "SADTXT", UNN
```

Creates a data file **SADTXT** on the default mass memory device, and copies the work file without line numbers into that file.

```
KEEP "ED", SAM
```

Creates a data file **ED** on volume **SAM**, and copies the work file to file **ED** in numbered format. If the data file **ED** already exists on volume **SAM**, the program displays:

```
ED, SAM already exists. Type "YES" to purge and then keep.  
PURGE?
```

If a **Y** or **YES** is entered, the data file is purged and the work file is copied to the file **ED** on volume **SAM**.

The LIST Command

```
{ L }  
LIST [Q][range][, OFFLINE]
```

The LIST command lists lines from the work file. Lines may be output to either the CRT or a printer.

If no parameters are specified, the current line is displayed on the CRT. If Q is not specified, the line number is not displayed with the line.

If a range is specified, all lines within the specified range are listed. Line numbers are not listed when Q is specified. The list operation may be terminated by pressing **HALT**.

If OFFLINE is specified, lines are printed on the default printer. The SET command may be used to select the offline printer, and to set the number of lines per page to be printed.

Some examples of this command are:

```
LIST ALL
```

Lists the entire work file on the display. The listing can be terminated at any time with **HALT**.

```
LISTQ 25/LAST
```

Lists all lines from line 25 to the display. No lines numbers are displayed.

```
L ALL, OFFLINE
```

Lists the entire work file to the default printer.

The MODIFY Command

`{ M
MODIFY } [range list]`

The MODIFY command modifies lines in the work file. The specified lines are displayed, one at a time, and the cursor is positioned to the right of the displayed line. The displayed line can then be modified and re-entered. The entire line may be replaced by pressing `CLEAR` and entering the new line.

The current line being modified may be re-displayed by pressing `CLEAR`, typing two slashes (`//`), and pressing `↑`. Pressing `HALT` before re-entering the line to be modified terminates the command and leaves the displayed line unchanged.

Some examples of this command are:

`MODIFY 5/6`

Displays lines 5 thru 6 for editing. Lines may be entered without modification, or may be modified before being entered.

`M FIRST`

Displays the first line of the work file for modification.

The SET Command

```
{ S } { LENGTH = nnn }  
{ SET } { PRINTER = n [, WIDTH=nnn] }  
        { LINES = nnn }
```

The set command is used to change EDITOR default parameters. The LENGTH parameter is used to set the maximum number of characters per line. The default length is 80 characters, but can be set from 20 thru 160 characters per line¹. Odd values are incremented, causing the length to always be even. The TEXT command automatically sets the length parameter when the UNNUMBERED option is not specified. The value of the length parameter is displayed by typing `CLEAR` Length `0`.

The PRINTER parameter is used to set the default printer for offline listings. The width is set to 132 characters per line, or is specified with the optional WIDTH parameter.

The LINES parameter is used to set the number of lines printed per page on offline listings. The default value is 66, and may be set to any integer value from 20 thru 999. The value of this parameter is determined by typing `CLEAR` Lp `0`.

Some examples of this command are:

```
SET LENGTH=160
```

Sets the maximum number of characters per line to 160. Lines longer than 160 characters are truncated and a warning message is displayed.

```
SET PRINTER=0
```

Sets the default printer (used for offline listings) to the standard printer. The width is set to 132 characters per line.

```
S LINES=88
```

Sets the number of lines printed per page to 88 for offline listings.

¹ If the work file is not empty, the length may only be increased from its current value.

The TEXT Command

`{ T
TEXT }` file spec `[,UNN
,UNNUMBERED]`

The TEXT command copies the specified data file into the work file. The old work file is lost. If the specified file is protected, the correct protect code must be entered before the file is copied into the work file.

If UNN or UNNUMBERED is specified, the lines are numbered as they are read. Lines longer than the length specified by the set command are truncated, and a warning message is displayed. If UNNUMBERED is not specified, the length parameter is automatically set, and lines are numbered using the line numbers appended to the end of each line.

Some examples of this command are:

```
T"SADTXT",UNNUMBERED
```

Copies the data file SADTXT from the default mass-memory device into the work file. Lines are automatically numbered as they are copied.

```
TEXT"ED,SAM"
```

Copies the numbered file ED from the volume SAM into the work file.

```
T"TFILE=F2,6,0"
```

Copies the numbered file TFILE from device F2,6,0 into the work file.

The WHILE Command

```
{
  N
}
{
  WHILE
}
```

The WHILE command repeats two command sequences. A command sequence can be up to two display lines containing EDITOR commands (separated by semicolons). When executed, WHILE prompts for two command sequences (each is entered with). After the second command sequence is entered, the command sequences are displayed and executed, one after the other, until either is pressed immediately after the command sequence is displayed, or until an error occurs. When is pressed during execution of an EDITOR command in the WHILE loop, it terminates the command and proceeds to the next command but does not terminate the WHILE loop. A WHILE command cannot be nested in another WHILE command.

An example of this command is:

```
FIND FIRST; WHILE
FINDQ"ABC"
MODIFY
```

This command sequence locates all lines containing the string ABC, and displays these lines for modification.

SCHEMA Definition

```
BEGIN DATA BASE base name [ , volume name ] ;
PASSWORDS:
    ucn password ;
    .
ITEMS:
    item name [sub-item count] type spec [ (control no. ) ] ;
    .
SETS:
    list of set definitions
END.
```

SCHEMA Parameters

base name – 1 thru 6 character data base name, beginning with a letter and containing uppercase letters, digits and dashes.

volume name – 1 thru 8 character string specifying a particular storage media. The name may not contain commas or semicolons.

ucn – A user class number. It is an integer from 1 thru 31.

password – A string of from 1 thru 8 characters not including semicolons. Imbedded blanks are removed.

item name – A 1 thru 15 character item name, beginning with a letter and containing letters, digits and dashes.

sub-item count – An integer from 1 thru 1022 (depending on item type) which specifies the replication count for the item whose type spec it precedes.

type spec – A specifier of the item type. It is either L, S, I or X. In the case of X, it is followed by an even integer from 2 thru 1022 specifying the string length.

control no. – An integer from 0 thru 127. This number may be retrieved by DBINFO. It is used by QUERY to determine the format used to print any data associated with that item.

Set Definition Syntax

Manual Master Set Definition

{NAME#} set name , **{MANUAL#}** (read list/write list) [, volume name] #
{N#}

{ENTRY#} item name (path count) #
{E#} item name #
:
item name #

{CAPACITY#} max entry count #
{C#}

Automatic Master Set Definition

{NAME#} set name , **{AUTOMATIC#}** (read list/write list) [, volume name] #
{N#} A

{ENTRY#} item name (path count) #
{E#}

{CAPACITY#} max entry count #
{C#}

Detail Data Set Definition

{NAME#} set name , **{DETAIL#}** (read list/write list) [, volume name] #
{N#} D

{ENTRY#} item name [(master set name)] #
{E#} item name [(master set name)] #
:
item name [(master set name)] #

{CAPACITY#} max entry count #
{C#}

Set Parameters

set name – A 1 thru 15 character set name, beginning with a letter and consisting of uppercase letters, digits and dashes.

read list – A list of user class numbers (including 0) separated by commas. The list may be null.

write list – A list of user class numbers (including 0) separated by commas. The list may not be null.

path count – An integer from 1 thru 8 corresponding to the number of paths between this master and the associated detail sets.

max entry count – An integer from 1 thru 65534 which specifies the maximum number of entries allowed in the set to which it pertains.

master set name – The name of a previously listed master data set.

DBML Statements and Advanced Access

Parameters

base\$ – A string variable which contains the data base name.

pass\$ – A string expression containing a left-justified string.

set – A numeric expression evaluating to a data set number.

set\$ – A string expression evaluating to a data set name.

mode – A numeric expression evaluating to a valid mode.

status – An integer array containing at least 10 elements in right-most dimension, used to return return codes on most DBML statements.

list\$ – A string expression evaluating to either “@”, “@:” or “@”. In all but the first case, any arbitrary character sequence may also follow.

buf\$ – A string variable, without any substring specifiers, which is used to transfer information between the BASIC program and the data base.

qual – A numeric expression evaluating to a valid item, set or volume number.

qual\$ – A string expression evaluating to a valid item, set or volume name.

item list – A list of string or numeric variables (or arrays) and SKPs which correspond to items in the data set specified in an IN DATA SET statement.

line list – A list of line numbers or labels which appears in an IN DATA SET ... USE REMOTE LISTS statement. Each line id must refer to an IN DATA SET LIST statement.

predicate\$ – a string variable returned by PREDICATE and used as the qualifier parameter by DBLOCK.

value – a string or numeric expression giving the value of the item to be locked.

item – a string expression specifying the data item within the set to be locked.

DBOPEN (base\$, pass\$, mode, status(*))

The DBOPEN statement opens the data base for access and defines the type of access allowed (i.e., read-only, exclusive, or shared).

DBCLOSE (base\$, { set } set\$, mode, status(*))

The DBCLOSE statement closes the data base and updates all information in the root file. A mode 4 close updates the root file, but leaves the data base open for future access. The set parameter is ignored.

```
DBGET (base$ {set } mode , status (*), list$ , buf$ {arg } )
```

The DBGET statement is used to retrieve information from the data base. If an IN DATA SET is active on the specified set, buf\$ will be unpacked into the appropriate variables.

```
DBUPDATE (base$ {set } mode , status (*), list$ , buf$ )
```

The DBUPDATE statement is used to modify values in an existing record in the data base (search item values cannot be modified). If an IN DATA SET is active on the specified data set, buf\$ will be updated with the current values of the appropriate variables before the operation.

```
DBPUT (base$ {set } mode , status (*), list$ , buf$ )
```

The DBPUT statement is used to add new entries to sets of type detail or manual. If an IN DATA SET is active on the specified set, buf\$ will be updated with the current values of the appropriate variables before the operation.

```
DBDELETE (base$ {set } mode , status (*))
```

The DBDELETE statement is used to remove entries from sets of type detail or manual.

```
DBFIND (base$ {set } mode , status (*), {item } {value } )
```

The DBFIND statement is used to find the head of a chain in a detail data set.

```
DBINFO (base$ {qual } mode , status (*), buf$ )
```

The DBINFO statement provides general information about the data base. The root file does not need to be on-line to obtain this information except when using the 400-series modes.

```
DBLOCK (base$ {set } mode , status (*))
```

The DBLOCK statement locks the entire data base or sections of the data base so modifications can be performed when the data base is open in shared mode.

```
DBUNLOCK (base$ {set } mode , status (*))
```

The DBUNLOCK statement unlocks a data base or section of the data base that was locked with a previous DBLOCK.

```
PREDICATE predicate$ FROM set1 [ , item1 [ , relop , value][ ; set2 ... [ ; setn]
```

The PREDICATE statement defines a section of the data base to be locked with the DBLOCK statement.

```
DBASE IS base$
```

The DBASE IS statement is used to specify the data base before any IN DATA SETs.

```
IN DATA SET { set  
              set$ } IN COM { USE ALL  
                             USE item list  
                             DIM ALL  
                             USE REMOTE LISTS line id list }
```

```
IN DATA SET { set  
              set$ } FREE
```

The IN DATA SET statement defines the automatic packing or unpacking procedure to be performed. Packing or unpacking of the buffer string is performed whenever a DBGET, DBUPDATE or DBPUT is executed on the specified data set (of the data base specified by the last DBASE IS statement). The FREE option allows the automatic packing and unpacking to be turned off.

The IN DATA SET LIST statement is a non-executable statement which is referenced by an IN DATA SET with the USE REMOTE LISTS option. This option is used when the USE list is too long to store as one program line.

Utility Statements

Parameters

base\$ – A string expression evaluating to the data base name.

maint\$ – A string expression evaluating to the maintenance password.

set list\$ – A string expression evaluating to a list of set numbers or names separated by commas. An * may be used, depending on the statement.

vol spec\$ – A string expression evaluating to a volume label or device specifier.

return var – A numeric expression to which the final execution status of the statement is assigned.

backup\$ – A string expression evaluating to the name of the backup file.

vol list\$ – A string expression evaluating to a list of backup volume names separated by commas.

```
DBCREATE base$ [ ; maint$ ] [ ; set list$ ] [ ; return var ]
                [ ; vol spec$ ]
```

The DBCREATE statement creates the data sets associated with a root file. Options are available for creating either all sets or only specific sets. A maintenance password may be specified for the first time DBCREATE is used to define a password. This password must be used on all subsequent accesses to the data base via utilities statements.

```
DBERASE base$ [ ; maint$ ] [ ; set list$ ] [ ; return var ]
                [ ; vol spec$ ]
```

The DBERASE statement erases all data sets or any group of data sets in the data base.

```
DBPURGE base$ [ ; maint$ ] [ ; set list$ ] [ ; return var ]
                [ ; vol spec$ ]
```

The DBPURGE statement purges either all data sets or any group of data sets in the data base.

```
DBSTORE base$ [ ; maint$ ] [ ; set list$ ] TO backup$ [ ON vol list$ ]
```

The DBSTORE statement backs up either all data sets or any group of data sets in the data base.

```
DBRESTORE backup$ [ ON vol$ ]
```

The DBRESTORE statement restores the data base using data stored in a backup file by a previous DBSTORE.

`DBPASS` root file spec , user-class number , old password `TO` new password

This binary statement changes the password for a stated user-class number.

`DBMAINT` root file spec , old word `TO` new word

This binary statement changes the maintenance password for a stated data base. The old word and new word parameters are string expressions from 0 through 16 characters, excluding nulls, spaces and rubouts. The old word specified must match the current maintenance word for the data base. The maintenance word is established when the root file is created via the `DBCREATE` statement.

`READ DBPASSWORD` file spec , maintenance word , string array variable

This binary statement reads all user passwords from the specified data base into a string array.

`WRITE DBPASSWORD` root file spec , maintenance word , string array variable

This binary statement re-assigns all passwords in the specified root file with those in a specified string array.

`XCOPY` source file spec , file type [,protect code]
`TO` dest file spec [,REPLACE]

This binary statement copies any file type except `SYST` and `DROM` from one volume to another. The source and destination file specs are string expressions containing the file name and, optionally, the volume spec. The file type is a string expression containing the four-digit file type (e.g., `DSET`, `ROOT`, or `BKUP`). If the destination file already exists, specify `,REPLACE` to copy the source file to the existing file. Error 851 occurs if the files are not compatible when `,REPLACE` is used.

APPENDIX D

IMAGE Comparison

The IMAGE/250 data base management system is modeled after IMAGE/3000. The following tables relate IMAGE systems in terms of capacities and capabilities. The most significant difference in terms of program structure is that IMAGE/250 only supports full record transfer. Packing and unpacking information into BASIC variables is handled automatically on the HP 250 via the IN DATA SET statement.

IMAGE Feature Comparison

	HP 250	HP 300	HP 3000
Items per data base	255	255	255
Data sets per data base	50	50	99
Data items per data set	127	127	127
Records per data set	65534	8388607	8388607
Paths (keys) from master to detail	8	16	16
# of characters in data set / item name	15	16	16
Security available	31 passwords	15 levelwords	63 passwords
Maximum media record length (bytes)	1024	2032	4096
Security level	set	set & item	set & item
Compound data items	yes	yes	yes
Sorted chains	no	no	yes
DBMS:			
DBOPEN modes	3	4	8
DBGET modes	4	5	8
DBINFO volume information	yes	no	no
DBLOCK:			
Data Base	yes	yes	yes
Group	no	yes	no
Data Set	yes	no	yes
Record	yes	no	yes
DBERROR	no	yes	yes
DBEXPLAIN	no	no	yes
Data transfer	entry	item	item
DBUS:			
Restructuring (load, unload)	yes	no	yes (minimal)
Ability to maintain single set	yes	no	no

IMAGE 250/300/3000 Modes¹

	Mode	250	300	3000	Description
DBOPEN	1	X	X	X	modify, allow concurrent modify (with locking)
	2			X	update, allow concurrent update
	3	X	X	X	modify, exclusive
	4			X	modify, allow concurrent read
	5			X	read, allow concurrent modify (with locking)
	6			X	read, allow concurrent modify
	7			X	read, exclusive
	8	X	X	X	read, allow concurrent read
	9		X		group lock
DBCLOSE	1	X	X	X	close data base
	2		X	X	close data set
	3	X	X	X	rewind data set
	4	X			update disc information
DBGET	1			X	re-read
	2	X	X	X	forward serial read
	3			X	backward serial read
	4	X	X	X	directed read
	5	X	X	X	forward chained read
	6		X	X	backward chained read
	7	X	X	X	calculated access
	8			X	primary calculated access
DBINFO	101	X	X	X	item position / security
	102	X	X	X	item description
	103		X	X	items in data base
	104	X	X	X	items in data set
	201	X	X	X	set position / security
	202	X	X	X	set description
	203	X	X	X	sets in data base
	204	x	X	X	data sets using item
	301	X	X	X	paths in data set
	302	X	X	X	primary search item
	401	X			data set volume number / status
	402	X			volume name
	403	X			volumes in data base / status
404	X			data sets on given volume	
DBLOCK	1	X	X	X	lock data base
	2	X	X	X	lock data base; no wait
	3	X		X	lock data set
	4	X		X	lock data set; no wait
	5	X		X	predicate lock
	6	X		X	predicate lock; no wait
	11	X			read lock data base
	12	X			read lock data base; no wait
	13	X			read lock data set
	14	X			read lock data set; no wait
	15	X			predicate read lock
	16	X			predicate read lock; no wait

¹ Only mode 1 is supported on each system for DBPUT, DBUPDATE, DBDELETE and DBUNLOCK.

APPENDIX E

IMAGE / 250 Error Messages

This appendix describes all error numbers and associated messages for the various systems comprising IMAGE / 250:

- Editor errors.
- Schema processor errors.
- Data base manipulation status errors.
- PACK and IMAGE / 250 execution errors.
- DBLOAD / DBUNLD errors.

In general, the EDITOR, SCHEMA, DBLOAD and DBUNLD utilities automatically translate each error code into an English-language description. If the file containing these descriptions is not available, however, only an error code is displayed.

EDITOR Error Messages

Error Code	Error Message
1	CLEAR NOT CONFIRMED, HOLD FILE UNCHANGED
2	CLEAR NOT CONFIRMED, WORK FILE UNCHANGED
3	FILE NOT FOUND
4	FILE NOT NUMBERED, WORK FILE IS EMPTY
5	FILE NOT NUMBERED, WORK FILE UNCHANGED
6	HOLD FILE FULL
7	ILLEGAL COMMAND
8	ILLEGAL FILE NAME
9	ILLEGAL LINE NUMBER
10	ILLEGAL SET PARAMETER
11	ILLEGAL SET PARAMETER VALUE
12	ILLEGAL VOLUME OR MASS MEMORY SPECIFIER
13	IMPROPER FILE TYPE
14	LINE ALREADY PRESENT
15	LINE NOT FOUND
16	LINE NUMBER OUT OF RANGE

- 17 NESTED WHILE COMMAND IS ILLEGAL
- 18 NO TEXT IN HOLD FILE
- 19 NO TEXT IN WORK FILE
- 20 NULL RANGE OR FIRST>SECOND
- 21 PURGE NOT CONFIRMED, TEXT NOT KEPT
- 22 SCRATCH FILE ERROR (FATAL)
- 23 STRING NOT FOUND WITHIN RANGE
- 24 SYNTAX ERROR
- 25 WORK FILE FULL...KEEP (NUMBERED) AND THEN TEXT
- 26 UNABLE TO OPEN OR READ FILE
- 27 UNDELIMITED FILE SPECIFIER
- 28 UNDELIMITED STRING
- 29 UNEXPECTED SYSTEM ERROR (FATAL)
- 30 VOLUME NOT FOUND
- 31 WARNING, COMMANDS FOLLOWING WHILE ARE LOST
- 32 WARNING, LINE TRUNCATED

IMAGE Status Errors

The following list describes the condition word values for IMAGE programming statements.

Condition Word	Error Description
----------------	-------------------

- | | |
|-----|---|
| 0 | Successful execution – no error. |
| -1 | No such data base.
Data base is currently opened in an incompatible mode.
Bad root file reference.
Data base opened exclusively. |
| -7 | Data base lock request was already made in current environment. |
| -10 | User may not open additional data bases, five are already opened. |
| -11 | Bad data base name or preceding blanks missing. |
| -12 | DBPUT, DBDELETE or DBUPDATE called with data base not locked. |

- 14 DBPUT, DBDELETE and DBUPDATE not allowed in access mode 8.
- 21 Bad password – grants access to nothing. Data item nonexistent or inaccessible.
Data set nonexistent or inaccessible. Data set volume nonexistent.
- 23 User lacks write access to data set.
- 24 DBPUT, DBDELETE, DBUPDATE not allowed on automatic master.
- 31 Bad mode.
DBGET mode 5 – specified data set lacks chains.
DBGET mode 7 – illegal for detail data set.
- 52 Item specified is not an accessible search item in the specified set.
Bad LIST variable – must be @Δ or @! or @. (Δ indicates blank)
- 91 Root file not compatible with current IMAGE/250 statements.
- 92 Data base requires creation.
- 94 Data or structure information lost. Data base must be erased or re-created.
- 95 No automatic master set entry for current detail. DBDELETE only.
- 96 Corrupt pointer value detected in current data set.
- 120 Not enough memory to perform DBLOCK.
- 122 Descriptor list bad. Not within string limits.
- 123 Illegal relational operator.
- 124 Descriptor too short; must be greater than or equal to 9 words.
- 125 Bad set name/ number.
- 126 Bad item name/ number.
- 127 Attempt to lock using a compound item.
- 128 Bad descriptor length for numeric item.
- 134 Two descriptors conflict.
- 135 Second lock is not allowed in modes 1,3,5,11,13 and 15.
- 136 Descriptor list exceeds 2047 words.
- 137 Qualifier parameter is of wrong type.
 - 11 End-of-file.
 - 12 Directed beginning of file.
 - 13 Directed end of file.
 - 15 End of chain.
 - 16 The data set is full.
 - 17 There is no chain for the search item value.
There is no entry with the specified key value.
No current record or the current record is empty.
The selected record is empty.
 - 18 Broken chain.
 - 20 Data base locked or contains locks.
Status word 3: 0 - data base locked. 1 - data set or entries locked.
 - 22 Data set locked by another process.

- 23** Entries locked within set.
- 24** Item conflicts with current locks.
- 25** Entry or entries already locked.
- 27** Relational operator type conflict.
- 41** DBUPDATE will not alter a search item.
- 43** Duplicate key value in Master.
- 44** Cannot delete a Master entry with non-empty Detail chains.
- 50** User's buffer is too small for requested data.
- 53** ARGUMENT field type incompatible with search field type (DBGET, mode 7, or DBFIND).
 ARGUMENT's current string length is less than the string length of the search field.
- 80** Data set volume is not on-line.
- 90** Root file volume is not on-line.
- 94** Corrupt data base opened successfully in mode 8.
- 1xx** There is no chain head for path xx.
- 3xx** The automatic master for path xx is full.
- 4xx** The master data set for path xx is not currently mounted (applies to DBPUT AND DBDELETE for detail data sets).

Status Array Contents Following an Error

The contents of the status array following an IMAGE/250 error (a non-zero condition word) is listed below.

Word	Description
1	Condition word
2 thru 4	Unchanged
5	0
6	Bits 0 thru 11: an id number from 401 thru 410 (see table below) Bits 12 thru 15: 0 or the mode value used to open the data base
7	Program line number
8	0
9	The mode parameter value
10	For HP use only

An identification number is associated with each DBML statement as shown below:

ID Number	DBML Statement
401	DBOPEN
402	DBINFO
403	DBCLOSE
404	DBFIND
405	DBGET
406	DBUPDATE
407	DBPUT
408	DBDELETE
409	DBLOCK
410	DBUNLOCK

PACK and IMAGE/250 Error Codes

Error Code	Error Description
200	Referenced line not a PACKFMT.
202	Insufficient dimension length in PACK statement, or insufficient current length in an UNPACK.
203	List item greater than 32K in PACK or UNPACK.
204	Conversion error.
205	UNPACK requires a source string of greater length.
210	Bad status array.
211	No DBASE IS statement active; improper data base specified or data base is not open.
212	Data set not found.
213	Excessive variables in list.
214	IN DATA SET already active for data set.
215	Number of elements does not match.
216	Variable type does not match with associated field in set.
217	String length in list insufficient, or length of list array greater than 255 bytes.
218	Variable not in common.
219	Line referenced is not an IN DATA SET LIST statement.
220	Improper or illegal use of maintenance word.
221	Data set not created. ¹
222	Needed volume lost during dismount.
223	Improper backup file.
224	Incomplete backup file.
225	Improper utility version number in root file.
226	Corrupt data base – must recreate it.
227	Corrupt data base – must erase it.
228	Data set cannot be restored without root file.
229	A null volume label exists on a data base or backup volume.
233	Required data set or root file not mounted.
320	Set or item specifier is out of range or is an invalid set or item name.
321	Relational operator is invalid.
322	The predicate specified is not a valid form.

¹ When executing DBCREATE, DBERASE or DBPURGE from the keyboard without a return variable, errors 54, 56, 64, 77 and 221 are not fatal. They are logged on the CRT along with information on the set to which the error pertains.

Schema Processor Messages

Warning Messages

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

COUNT HAS BAD FORMAT

The number following `ERRORS=` is not an integer from 0 thru 999, or the number following `LINES=` is not an integer from 20 thru 999 in the `$CONTROL` command.

ILLEGAL COMMAND

The command statement is not a `$PAGE`, `$TITLE`, or `$CONTROL` command, or a parameter list is not separated by commas.

IMPROPER COMMAND PARAMETER

The command parameter specified in the `$CONTROL` statement is not valid.

MISSING QUOTATION MARK

The character string specified in a `$PAGE` or `$TITLE` must be bracketed by quotation marks.

Error Messages

The following messages do not halt schema processor operation; 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 search item.

BAD CAPACITY OR TERMINATION

Either the number in the CAPACITY: statement is not an integer between 1 and 65534 or a semicolon is missing.

BAD DATA BASE LABEL OR TERMINATOR

The specified data base label is null or has more than 8 characters.

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 thru 6 alphanumeric characters beginning with an alphabetic, or it is not followed by a comma or a semicolon.

BAD DATA SET TYPE

Data set type designator is not AUTOMATIC (or A), MANUAL (or M), or DETAIL (or D).

BAD ITEM FORMAT OR DELIMITER

The item control number is not an integer from 1 thru 127, or the format number is not enclosed in parenthesis.

BAD ITEM LENGTH OR TERMINATOR

The string item length is not an integer from 1 thru 1022.

BAD ITEM TYPE DESIGNATOR

The item type designator is not I, S, L, or X.



BAD PASSWORD WORD OR TERMINATOR

The password is not terminated by a semicolon.

BAD PATH COUNT OR TERMINATOR

The path count of the master set is not an integer from 0 thru 8 (or 1 thru 8 for automatic masters), or the path count is not enclosed in parenthesis.

BAD PATH SPECIFIER DELIMITER

No path specifier is present, or the path count is not enclosed in parenthesis.

BAD READ PASSWORD OR TERMINATOR

The read-only user class number is not an integer from 0 thru 31, or is not followed by a comma or slash.

BAD READ/WRITE SPECIFICATION DELIMITER

The read/write specifiers are not enclosed in parenthesis.

BAD SET LABEL OR TERMINATOR

The set volume label specifier is null or is longer than eight characters.

BAD SET NAME OR TERMINATOR

The data set name is not 1 thru 15 alphanumeric characters beginning with an alphabetic, or is not followed by a comma.

BAD SUB-ITEM COUNT OR TERMINATOR

The sub-item count for the data item defined is not an integer from 1 thru 255.

BAD TERMINATOR – ‘;’ or ‘,’ EXPECTED

Items within an entry definition must be terminated by commas. The last item in the entry definition must be terminated with a semicolon.

BAD TERMINATOR – ‘;’ EXPECTED

The BEGIN DATA BASE line, item definition line, or NAME: line is not terminated by a semicolon.

BAD WRITE PASSWORD OR TERMINATOR

The read/write user class number is not an integer from 0 thru 31, or is not followed by a comma or a right parenthesis.

‘BEGIN DATA BASE’ EXPECTED

A BEGIN DATA BASE line must be the first, non-command 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 the 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: section declaration lines must each be on separate lines.

ILLEGAL ITEM NAME OR TERMINATOR

The data item name is not 1 thru 15 alphanumeric characters beginning with an alphabetic, or is not followed by a comma.

ILLEGAL PASSWORD NUMBER

The password user class number is not an integer from 1 thru 31, or the number has been used in more than one password definition.

ITEM LENGTH NOT INTEGRAL WORDS

The character count for a string item must be even.

ITEM LENGTH TOO LONG

The data item length exceeds 1022 bytes.

MASTER DATA SET LACKS EXPECTED DETAILS

A master data set was defined with a non-zero path count, but the number of detail search items which back-referenced the 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 word, 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 search item is another detail set rather than a master data set.

SEARCH ITEMS NOT SIMILAR

Master search item must be of the same type and length as the related detail data set's search item.

SEARCH ITEM NOT SIMPLE

All data items defined as a search item must have a sub-item count of 1.

SET HAS NO PATHS AVAILABLE

More detail data set search items have back-referenced a master data set than had been specified by the master set path count.

SET TOO LARGE

The product of the media record length and the set capacity exceeds 16.7767 Mbytes (i.e., the space required to store the set exceeds 65534 sectors).

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 50 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 23 different set volume labels may be defined.

TOO MANY PATHS IN A DATA SET

No more than eight search 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 search item was not previously defined in the set part of the schema.

Fatal Error Messages

These messages indicate that Schema Processing has been terminated.

DUPLICATE ROOT FILE NAME

A file on the destination device already contains a file with the same name as the root file to be created.

FATAL ERROR error number ENCOUNTERED – status

Schema encountered a program error while processing. The error number corresponds to the error encountered. The status number is for HP use only.

FATAL FILE ERROR error number ENCOUNTERED

Schema encountered a file error during processing. The error number corresponds to the error encountered.

INCOMPATIBLE PROGRAM FILE SCHOV2

The revision code of the SCHOV2 program file does not match the revision code of the SCHEMA program.

INCOMPATIBLE PROGRAM FILE SCHOV3

The revision code of the SCHOV3 program file does not match the revision code of the SCHEMA program.

INCOMPATIBLE VERSION OF SCHERR FILE

The revision code of the SCHERR error file does not match the current revision code of the SCHEMA program.

INSUFFICIENT SPACE FOR SCRATCH FILE

The mass storage device containing the Schema Processor must contain 120 sectors of contiguous file space for the search file. Purge any temporary files and run the program again.

NUMBER OF ERRORS EXCEEDS MAXIMUM

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.

PROGRAM TERMINATED

The HALT or EXIT key was pressed during processing. No ROOT file was created.

UNABLE TO LOAD FILE 'SCHOV2'

SCHEMA was unable to load the program file SCHOV2 during processing. No ROOT file was created.

UNABLE TO OPEN ERROR FILE SCHERR

SCHEMA was unable to find or open the SCHERR error file.

UNABLE TO LOAD FILE 'SCHOV3'

SCHEMA was unable to load the program file SCHOV3 during processing. No ROOT file was created.

UNABLE TO OPEN ROOT FILE – error number

SCHEMA was unable to create or open the root file. The error number corresponds to the error encountered by SCHEMA.

UNABLE TO OPEN SCRATCH FILE – error number

SCHEMA unable to create or open the \$SCH\$x scratch file. The error number corresponds to the error encountered by SCHEMA. The x is the user id number currently running the Schema Processor.

UNABLE TO OPEN TEXT FILE – error number

SCHEMA unable to open or read the specified data file containing the schema definition. The error number corresponds to the error encountered by SCHEMA.

UNEXPECTED EOF ON TEXT FILE

The schema text file contained an end-of-file before an END line was encountered.

DBLOAD/DBUNLD Error Messages

Error Number	Error Message
1	INCORRECT PASSWORD The specified maintenance password does not match the data base maintenance password.
2	IMPROPER SET COUNT¹ The number of data sets in the data base is out of range.
3	IMPROPER ITEM COUNT¹ The number of items in the current data set is out of range.
4	SEARCH ITEM SUBCOUNT >1¹ The sub-item count of the search item of the current data set is greater than one.
5	UNKNOWN SEARCH ENTRY TYPE¹ The search item type is not INTEGER, SHORT, REAL or STRING.
6	IMPROPER SEGMENT ENTRY COUNT A program or system failure has caused the creation of a data set backup segment to fail.
7	PROGRAM COMPLETION REQUIRES ROOT FILE².
8	NO ROOM ON CURRENT BACKUP VOLUME There is no free space on the specified backup volume to create the backup file.
9	DATA SET NAME NOT FOUND The specified data set name is not in the data base.
10	DATA BASE STATUS status A data base operation has failed, producing the status information shown.
11	DATA BASE NOT AVAILABLE The data base cannot be opened for exclusive access.
12	BACKUP FILE VOLUMES OUT OF ORDER The backup segment on the backup volume does not correspond with a previous segment.

¹ Indicates that data or structural information within the data base has been lost, preventing the operation from completing.

² This message is for information or warning to the user. Program execution will continue.

13 DUPLICATE BACKUP FILE NAME²

A file with the backup file name on the backup volume must be purged before the backup file may be created.

14 PURGE NOT CONFIRMED; OLD FILE KEPT

The response to the 'purge file' request was 'N' or 'NO'. The original file is unchanged.

15 FATAL ERROR error ENCOUNTERED IN PROGRAM program name--status

The named program encountered a program error while processing. The error number corresponds to the error encountered. The status number is for HP use only.

16 ROOT FILE NOT FOUND

The data base root file does not exist on the specified volume.

17 ATTEMPT TO UNLOAD OR LOAD AUTOMATIC MASTER

The single data set option was used to request an unload or load of an automatic master data set.

18 ITEM POSITION VALUE EXCEEDS ITEM COUNT

An entry in the backup set-item-position list exceeds the number of items in the backup data set.

19 IMPROPER VOLUME COUNT¹

The number of data base volumes is out of range.

20 ITEM TYPES DO NOT MATCH

The item types of the backup data set (possibly restructured with the 're-order' option) do not match the item types of the destination data set.

21 ATTEMPT TO LOAD CORRUPT DATA BASE

The data base has been marked corrupt by IMAGE; the data base must be erased before it is loaded.

22 REQUESTED DATA SET NUMBER NOT FOUND

The source data set number is not in the backup file.

23 ZERO LENGTH BACKUP FILE

Directory information on the backup volume is inconsistent.

24 IMPROPER DATA SET NUMBER¹

The data set number of the specified data set name is 0.

¹ Indicates that data or structural information within the data base has been lost, preventing the operation from completing.

² This message is for information or warning to the user. Program execution will continue.

- 25 FORM IS NOT COMPLETE**
All of the necessary values have not been entered: the cursor is positioned in the required field.
- 26 FILE NAME NOT FOUND**
The backup file name is not on the backup volume.
- 27 IMPROPER PATH NUMBER¹**
The path number is out of range.
- 28 IMPROPER INPUT VALUE**
An input value is invalid or out of range; the cursor is positioned at the improper value.
- 29 INCORRECT FILE TYPE**
For DBUNLD, the indicated file cannot be purged. For DBLOAD, the specified file is not a backup (BKUP) file.
- 30 BACKUP FILE NOT CREATED BY DBUNLD UTILITY**
The internal format of the backup file is incorrect. The backup file may have been created by another backup utility.
- 31 ERASE REQUIRES ALL VOLUMES BE MOUNTED²**
The data base is marked corrupt and all data base volumes must be mounted for the data base to be erased.
- 32 FEWER ENTRIES UNLOADED THAN EXPECTED**
The number of data entries retrieved from the data set is less than the correct number of entries in the data set.
- 33 FEWER ENTRIES LOADED THAN EXPECTED**
The number of entries in the backup data set segment is less than the anticipated number of data set entries.
- 34 DATA BASE IS MARKED CORRUPT²**
A data base marked corrupt by IMAGE has been opened to allow the recovery of data from the data base.
- 35 PROGRAM FILE VERSION DISAGREEMENT**
The revision code of the program (segment) loaded does not agree with the previous program segment revision code.
- 36 BACKUP SET NUMBER NOT IN DATA BASE**
A data set number in the backup file does not exist in the data base.

¹ Indicates that data or structural information within the data base has been lost, preventing the operation from completing.

² This message is for information or warning to the user. Program execution will continue.

- 37 READ FAILURE IN DATA SET RECORD POSITION** number
A mass memory read failure has occurred for the data set record position shown. The unload process will continue with the next data set record position.
- 38 SEARCH ITEM ERROR**¹
Data base search item information is inconsistent.
- 39 DATA SET ENTRY OMITTED FOR SEARCH VALUE** value²
For a master set: The manual-master entry for the search value shown is duplicate and cannot be added to the data set.
For a detail set: The related manual-master entry for the search value shown is missing and the detail entry with this search item value cannot be added to the data set. The load process will continue with the next entry.
- 40 VOLUME NAME TO LONG: TRUNCATED VALUE** name²
The specified volume name is longer than eight characters. The first eight characters of the name will be used.
- 41 FILE PROTECT CODE DOES NOT MATCH**
The specified protect code does not match the backup file protect code.
- 42 MISSING DATA SET** number
The data set number has not been created. If this message is displayed during a data base erase, no number will be displayed.
- 43 DATA ITEM LENGTH OR PRECISION LOST**²
During data base restructuring, either non-blank characters were lost from the end of a string, or significant digits or exponent range was lost in a numeric conversion.
- 44 ITEM CONVERSION ERROR**¹
Data base or backup file item-length information is incorrect.
- 45 CORRUPT DATA BASE REQUIRES SERIAL MODE**
Chained mode unload is not allowed on the data base. Serial mode operation must be used to access the data entries.
- 46 DATA SET REQUIRES ITEM RESTRUCTURING**²
Item conversions must be performed on the backup file data entries to load the entries into the data set. Numeric value conversions or string length conversions are required.

¹ Indicates that data or structural information within the data base has been lost, preventing the operation from completing.

² This message is for information or warning to the user. Program execution will continue.

APPENDIX F

Hash Algorithm Description

Introduction

The calculated access method for IMAGE master data sets is supported by a hashing algorithm. IMAGE takes a key item value and calculates a logical record number from it. Different keys that map to the same logical record number are called synonyms and are linked together in a synonym chain. Use of the STANDARD key transformation (hashing algorithm) will enhance the speed at which data can be accessed if ASCII search keys are used.

PRE-OS6

Operating systems B.05 and prior use the hashing algorithm PRE-OS6. Whatever the data type of the key value, the hashing algorithm treats it as a sequence of 16 bit integers or words. The words in the search key are combined in a procedure known as Exclusive OR. This 16 bit result is then rotated to the right one bit position, with the least significant bit rotated around to the most significant bit position. This value is then divided by the data set capacity and the remainder of this division forms the logical record number for this key value. The following example uses a search key 3 words long and a data set capacity of 101.

Example:

Word Count = 3

Hash Total	00000000	00000000
“HP”	01001000	01010000

Hash Total	01001000	01010000
“25”	00110010	00110101

Hash Total	01111010	01100101
“0”	00110000	00100000

Hash Total	01001010	01000101
------------	----------	----------

rotate right 1 bit	10100101	00100010
--------------------	----------	----------

convert to unsigned 16 bit integer	42274
---------------------------------------	-------

mod capacity or 101 = logical record number	56
--	----

Standard

The current version of the hashing algorithm adds an extra step in the Exclusive OR procedure. Every other word is rotated right 4 bits. The odd numbered words (1, 3, 5...) are rotated if the length of the search key is even. The even numbered words (2, 4, 6...) are rotated if the length of the search key is odd. This has the effect of randomizing every bit position so that maximum randomness is obtained. The following example uses a search key 3 words long and a data set capacity of 101.

Example:

Word Count = 3

Hash Total	00000000	00000000
"HP"	01001000	01010000
Hash Total	01001000	01010000
"25"	01010011	00100011*
Hash Total	00011011	01110011
"0"	00110000	00100000
Hash Total	00101011	01010011
rotate right 1 bit	10010101	10101001
convert to unsigned 16 bit integer	38313	
mod capacity or 101 = logical record number	34	

*Because the word count was odd, the second word of the search key was rotated. The bit pattern for "25" is 00110010 00110101. The rotated result is in the line above.