HP 3000 Computer Systems

HEWLETT hp PACKARD

IMAGE
Data Base Management System
reference manual

HP 3000 Computer Systems

IMAGE
Data Base Management System

Reference Manual

HEWLETT hp PACKARD

5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA 95050

Part No. 32215-90003
Product No. 32215B

Printed in U.S.A. 4/78
Changed: 9/78

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 Company.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and the dates when pages were changed in updates to that edition. Within the manual, any page changed since the last edition has the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. When an update is incorporated in a subsequent reprinting of the manual, these bars are removed.

First Edition. . . . . Apr 1978

Changed Pages	Effective Date	Changed Pages	Effective Date
iii thru v	Sept 1978	5-40 thru 5-42	Sept 1978
vii thru x	Sept 1978	5-49 thru 5-50	Sept 1978
2-3 thru 2-5	Sept 1978	5-54	Sept 1978
2-16	Sept 1978	6-1 thru 6-6	Sept 1978
3-5 thru 3-6a	Sept 1978	6-8	Sept 1978
3-18	Sept 1978	6-14	Sept 1978
3-21 thru 3-23	Sept 1978	6-16 thru 6-16e	Sept 1978
4-1 thru 4-6	Sept 1978	6-18 thru 6-29	Sept 1978
4-8	Sept 1978	7-3a thru 7-4	Sept 1978
4-10 thru 4-21	Sept 1978	7-6	Sept 1978
4-23 thru 4-25	Sept 1978	8-10	Sept 1978
4-29 thru 4-34	Sept 1978	A-9 thru A-12	Sept 1978
4-37 thru 4-50	Sept 1978	A-14 thru A-15	Sept 1978
5-9	Sept 1978	A-17 thru A-22a	Sept 1978
5-11 thru 5-14	Sept 1978	A-25 thru A-26	Sept 1978
5-25 thru 5-25a	Sept 1978	C-2	Sept 1978
5-29 thru 5-32	Sept 1978	D-1 thru D-2	Sept 1978

PRINTING HISTORY

New editions are printed when extensive changes are made to the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The edition printing date on the title page and back cover of the manual changes only when a new edition is published. When the manual is reprinted, the current update, if any, is incorporated into the manual. No new content is added to the manual; neither the printing date on the title page nor back cover, nor the edition number change. The date of the incorporated printing is added to the back cover.

The software product part number printed beside the date indicates the version and update level of the software product at the time the manual edition or update was issued. Many product updates and fixes do not require manual changes, and conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

First Edition	Apr 1978	32215B.00.00
Update #1	Sep 1978	32215B.01.00

PREFACE

This manual describes the IMAGE/3000 Data Base Management System for HP 3000 computers*. It is the reference document for all persons involved in designing and maintaining a data base and the applications programmers who write programs to access a data base.

Designers of IMAGE data bases will find knowledge of the HP 3000 Multiprogramming Executive (MPE) operating and file systems useful in determining the amount of system resources, such as disc space and computation time, needed to maintain a specific data base. Because access to IMAGE data bases requires the use of a host programming language, applications programmers need familiarity with at least one of the programming languages available on the HP 3000 computer: COBOL, FORTRAN, SPL, BASIC, or RPG.

In addition to this manual, you may need to consult the following manuals:

Manual	Part Number
BASIC/3000 Compiler Reference Manual	32103-90001
BASIC Interpreter Reference Manual	30000-90026
COBOL/3000 Reference Manual	32213-90001
Console Operator's Guide	30000-90013
EDIT/3000 Reference Manual	03000-90012
Error Messages and Recovery Manual	30000-90015
FORTRAN Reference Manual	30000-90040
General Information Manual	30000-90008
MPE Commands Reference Manual	30000-90009
MPE Intrinsics Reference Manual	30000-90010
QUERY Reference Manual	30000-90042
RPG/3000 Compiler Reference and Application Manual	32104-90001
System Manager/System Supervisor Manual	30000-90014
Systems Programming Language Reference Manual	30000-90025
DS/3000 Reference Manual	32190-90001
Machine Instruction Set Manual	30000-90022

*For 3000 systems which are operating with MPE-C, refer to the *IMAGE Reference Manual* (30000-90041).

CONVENTIONS USED IN THIS MANUAL

NOTATION — **DESCRIPTION**

[ ]

An element inside brackets is *optional*. Several elements stacked inside a pair of brackets means the user may select any one or none of these elements.

Example: $\begin{bmatrix}A\\B\end{bmatrix}$ user may select A or B or neither

{ }

When several elements are stacked within braces the user **must** select one of these elements.

Example: $\begin{Bmatrix}A\\B\\C\end{Bmatrix}$ user must select A or B or C.

italics

Lowercase italics denote a parameter which must be replaced by a user-supplied variable.

Example: CALL *name*
name one to 15 alphanumeric characters.

underlining

Dialogue: Where it is necessary to distinguish user input from computer output, the input is underlined.

Example: NEW NAME? ALPHA1

superscript C

Control characters are indicated by a superscript C

Example: Y^C

return

return in italics indicates a carriage return

linefeed

linefeed in italics indicates a linefeed

. . .

A horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.

upper case

Words in upper case appearing in format statements must appear exactly as shown.

Example: SETS:

CONTENTS

CONTENTS (continued)

Section V Page
LANGUAGE CONSIDERATIONS AND EXAMPLES

Section VI Page
CREATING AND MAINTAINING THE DATA BASE

Section VII Page

INTERNAL STRUCTURES AND TECHNIQUES

Section VIII Page

USING A REMOTE DATA BASE

Appendix A Page

ERROR MESSAGES

Appendix B Page

Appendix C Page

Appendix D Page

ILLUSTRATIONS

TABLES

INTRODUCTION

SECTION I

IMAGE is a set of programs and procedures which you can use to define, create, access, and maintain a data base.

WHAT IS A DATA BASE AND WHY USE ONE?

A data base is a collection of logically-related files containing both data and structural information. Pointers within the data base allow you to gain access to related data and to index data across files.

The primary benefit derived from use of the IMAGE data base management system is time savings. These savings are typically manifested in the following areas:

- File Consolidation

  Most information processing systems that serve more than one application area contain duplicate data. For example, a vendor's name may appear in an Inventory File, an Accounts Payable File, and an Address Label File.

  The data stored in these three files probably varies slightly from file to file, resulting not only in wasted file space but also inconsistent program output. Redundant and inconsistent information severely dilutes any system's capacity to deal with large amounts of data.

  File consolidation into a data base eliminates most data redundancy. Through the use of pointers, logically related items of information are chained together, even if they are physically separated. In the example of vendor names and addresses, only one set of data would be stored. Through the use of logical associations, the data could be used by any program needing it. Since there is only one record to retrieve and modify, the work required for data maintenance is greatly reduced. Finally all reports drawn from that item of information are consistent.

- Program File Independence

  Conventional file structures tend to be rigid and inflexible. The nature of conventional file management systems require that the logic of application programs be intricately interwoven with file design. When it becomes necessary to alter the structure of a file, a program must be written to change the file, and programs that access the file must be changed to reflect the file change. Since change is the rule rather than the exception in data processing, a large percentage of total time and manpower is spent reprogramming.

  IMAGE allows the data structure to be independent of the application program. Data item relationships are independently defined. Changes in the data base structure need only be incorporated into those programs that manipulate the changed data. User programs need view only that portion of the data base description that pertains to each program's processing requirements. Since all references to the data base are resolved at execution time, only those programs affected by changes to the data base description need be changed.

- Versatility

  Conventional file organization techniques allow limited access to the data they contain. Most structures allow single key access with additional relational access available only through the implementation of extensive application level programming support.

  IMAGE allows data to be accessed with multiple keys as well as through a variety of other access methods.

- Rapid Retrieval

  Conventional file organization frequently requires the use of multiple file extracts, sorts and report programs to produce meaningful output data across file boundaries. One-time information requests frequently require weeks to implement, during which time the usefulness of the requested data may have eroded considerably.

  QUERY, HP's Data Base Inquiry Facility, or user-written inquiry programs which use the IMAGE procedures, allow instant interrogation of the data base by individuals with access to the system.

- Data Security

  Conventional file management systems contain extremely limited data security provisions. Access to computer readable data may be denied to individuals with system access only by providing physical protection for the media upon which the file is stored; for example, the use of a data vault for storage of sensitive data stored on magnetic tape or disc.

  IMAGE provides security at the account, file group, and data element level. The implementation of security at the item level allows sensitive data to be stored on-line under the control of IMAGE, a data base manager or designer, and system manager, with minimal regard for additional security provisions. IMAGE security provisions can limit even programmer or operator access to extremely sensitive information.

While implementing a new application system, IMAGE can be expected to save time in the following ways:

- Program Development

  The data base structure can be defined and built without the use of special purpose application level programming. Since control of the linkage portion of the data base is under IMAGE software control, the programmer need not be concerned with testing the structure and can concentrate on the functional programming task at hand. If available, QUERY can be used to build test data as well as to interrogate the results of program and system tests. This feature eliminates the requirement that file-related programs be completed before meaningful functional programs can be written. It is no longer necessary to hold up functional program testing until file building or file maintenance programs are completed. In this manner, more modules of a given system can be tested in parallel.

  A specific benefit in the COBOL environment is in the area of program coding time. The programmer need only define File Division entries for those files which exist outside the control of IMAGE. Typically, such files are concerned with original entry into the processing cycle (data entry files) and with report files. All data under the control of IMAGE is implicitly defined in every program which accesses the data base. The programmer need not code the data division entries associated with anything except the detail data used by a given program.

The time-savings generated in correct data definition the first time the program is coded, as well as in the correct description of the physical location of the data to be processed, will reap significant benefits in the program test cycle.

- Program Maintenance

  Throughout the life of a system, processing requirements evolve as the usefulness of the data is explored. As file organization concepts change with the needs of the application, some data restructuring can be done with little impact on existing programs. Changes to the structure of an existing data base affects only those programs that process the changed data; no other programs in the system need be recompiled to reflect the new data base structure.

  The evolution of the data base is not limited by the need to balance the cost of changing an existing system against the benefits to be derived from the new structure. It is not necessary to do a "where-used" evaluation on a data item carried in multiple files to assess the impact of a data change on existing systems.

  Finally, the accessibility of data need not be limited by design decisions made during initial system design. The structure of a data base can evolve with the needs of the application user. The application designer no longer has to attempt to anticipate the needs of the user across the full life of the system.

- Special Information Needs

  The requirement for one-time information in a format that has never been requested before is no longer the bane of data processing users. The user with a special data requirement can get to any subset of information on the data base, frequently without the intervention of a programmer.

  Volatile analytical data requirements can be filled in a minimal amount of time by the people who need the data. The time savings in programming overhead and report specification generation can be enormous.

In summary, effective use of IMAGE can remove a large portion of the overhead associated with integrated system design from the shoulders of application analysts and programmers. It affords the opportunity to channel system design talents into functional rather than structurally-supportive design tasks.

HOW TO USE IMAGE

Figure 1-1 summarizes the use of IMAGE in the following steps:

① DESIGN OF THE DATA BASE

A data base designer (system analyst) or team of designers determine what data is required by all the application projects that will share the data base. They determine which data should be protected from unauthorized access and how the data will be used. These design considerations and others described in Appendix C determine the data base content and structure.

② DESCRIPTION OF THE DATA BASE

Once the design is complete it is described using the IMAGE data base description language. This external description is called a *schema*. The data base creator processes the schema using the IMAGE Schema Processor which creates an internal description of the data base called a *root file.* Section III contains the description language syntax and operating instructions for the Schema Processor.

③ CREATION OF THE DATA BASE FILES

DBUTIL, an IMAGE utility program, builds the data base files according to requirements of the data base structure specified in the root file. The files contain no data initially.

④ STORAGE AND RETRIEVAL OF THE DATA

IMAGE provides a set of library procedures which can be called from COBOL, FORTRAN, SPL, or BASIC language application programs. The data base can also be used with RPG programs but the Report Program Generator issues the calls to IMAGE procedures. The application project members can design and write programs in the programming language which best suits their needs and call the IMAGE procedures to store, modify, retrieve, and delete data. These procedures rapidly locate the data, maintain pointer information, manage the allocated file space, and return status information about the activity requested. Each procedure is described in detail in Section IV and examples of calling them from the different languages are given in Section V.

⑤ MAINTENANCE OF THE DATA BASE

The IMAGE utility programs may be used to maintain backup copies of the data base and perform other utility functions such as restructuring the data base. These programs are described in Section VI. You may also use the IMAGE procedures to write your own maintenance programs.

HOW TO USE THIS MANUAL

The information in this manual is presented in the order in which you will begin to use the various IMAGE modules. A text discussion of the overall purpose of a module and definitions of terms used to describe the module precede the reference specifications which are identified by large headings to enable you to locate them easily.

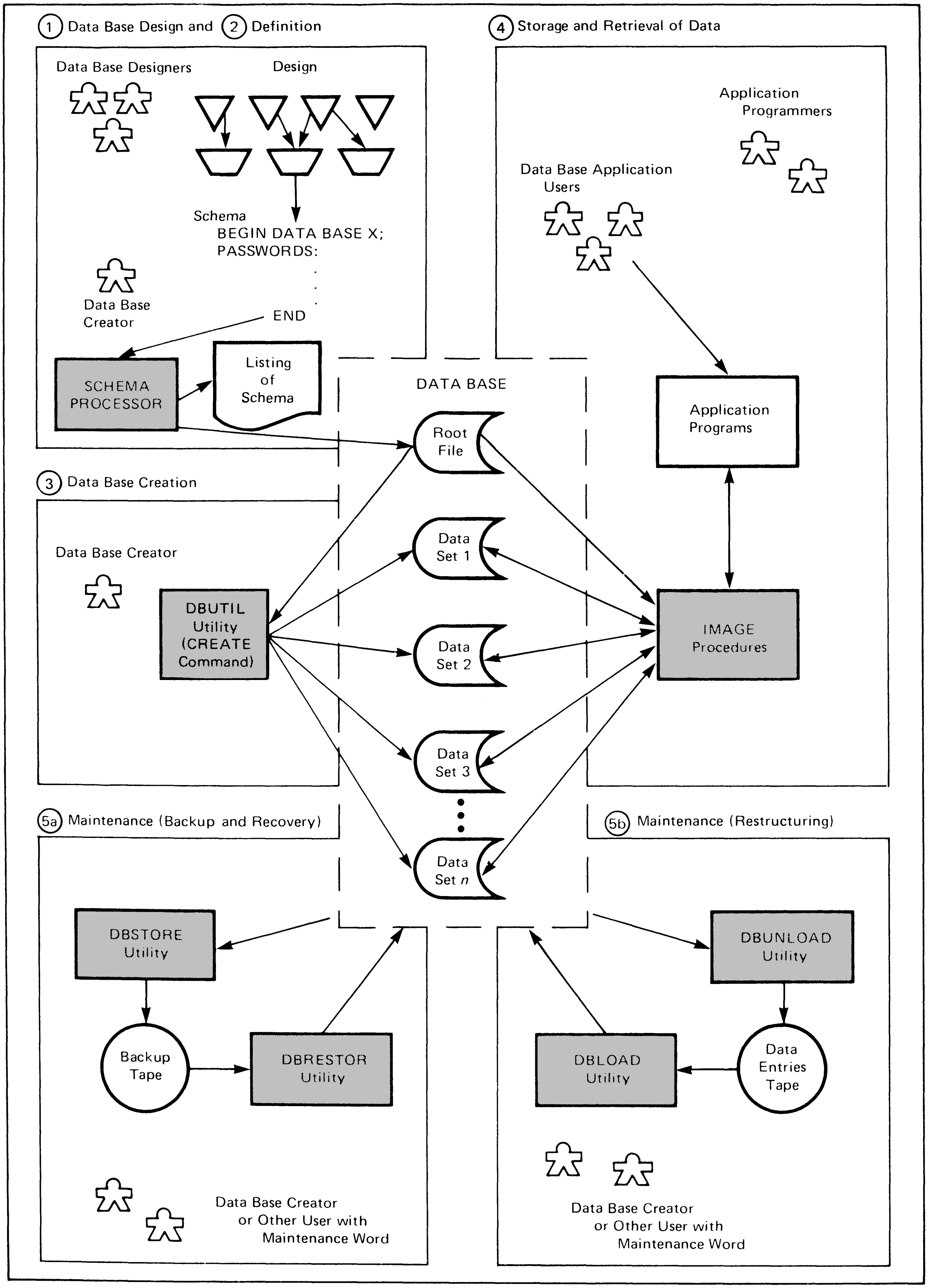


Figure 1-1. IMAGE Overview

Each section assumes a knowledge of the material presented in preceding sections. Therefore, it is recommended that you read the manual the first time from beginning to end, possibly skipping the discussion of topics which are already familiar to you.

The internal structure of IMAGE elements and methods used to perform certain functions are presented in the last section, Section VII. This section can be referenced at any time if you want to know exactly how something is accomplished by IMAGE, but it is not necessary to understand the material in this section to use IMAGE.

If your system has Distributed System (DS/3000) capability, refer to Section VIII for information about accessing a data base residing on another HP 3000 system.

Appendix A contains a description of the error messages issued by the various IMAGE modules and Appendix B provides additional information about sharing the data base. A summary of important considerations when designing the data base is provided in Appendix C.

The conventions used in this manual are described on page vi.

DATA BASE PERSONNEL

The terms data base manager, data base creator, and data base designer may refer to one or more persons. Designer refers to anyone who cooperates in the design of the data base. The creator is defined by the MPE user name, account, and group used when executing the Schema Processor to create the root file and when executing the DBUTIL program to create the data base files. The data base manager is responsible for coordinating data base use. This person knows the passwords and can authorize others to use the data base by making a password available if it is needed for a particular application. The data base manager is also responsible for system backup and recovery. The data base creator and manager may be the same person. If not, the manager will probably have access to the user name and account in which the data base resides or to the maintenance word which is defined in Section VI.

DATA BASE STRUCTURE AND PROTECTION

SECTION II

An understanding of the data base structure is necessary before the data base can be designed. This section describes the various data elements and their relationships

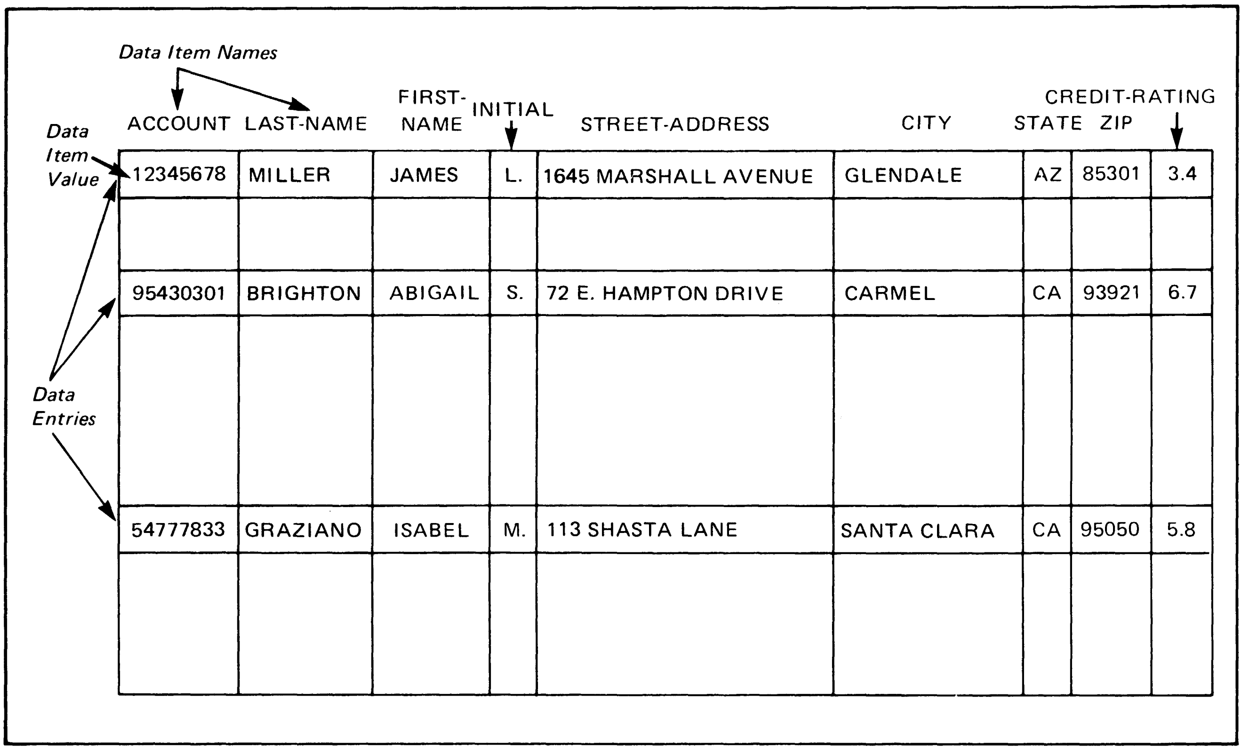
DATA ELEMENTS

A data base is a named collection of related data. It is defined in terms of data items and data sets. Figure 2-1 contains a sample of one data set from a data base named STORE which will be used as an example throughout this manual. The data set is named CUSTOMER. The information in this data set pertains to the customers of a business. All the data about a particular customer is contained in a *data entry.* Each piece of information such as account number or last name is a *data item.*

DATA ITEMS

A data item is the smallest accessible data element in a data base. Each data item consists of a value referenced by a data item name, typically selected to describe the data value. In general, many data item values are referenced by the same data item name, each value existing in a different data entry.

For example, in figure 2-1, the data item FIRST-NAME has the value JAMES in one data entry and ABIGAIL in another data entry.



ACCOUNT	LAST-NAME	FIRST-NAME	INITIAL	STREET-ADDRESS	CITY	STATE	ZIP	CREDIT-RATING
12345678	MILLER	JAMES	L.	1645 MARSHALL AVENUE	GLENDALE	AZ	85301	3.4
95430301	BRIGHTON	ABIGAIL	S.	72 E. HAMPTON DRIVE	CARMEL	CA	93921	6.7
54777833	GRAZIANO	ISABEL	M.	113 SHASTA LANE	SANTA CLARA	CA	95050	5.8

Figure 2-1. CUSTOMER Data Set Sample

COMPOUND DATA ITEMS. A compound data item is a named group of identically defined, adjacent items within the same data entry. Each occurrence of the data item is called a sub-item and each sub-item may have a value. A compound item is similar to an array in programming languages such as FORTRAN and BASIC. A data entry might contain a compound item named MONTHLY-SALES with 12 sub-items in which the total sales for each month are recorded. (If you plan to use QUERY, avoid using compound data items.)

DATA TYPES. The data base designer defines each data item as a particular type depending on what kind of information is to be stored in the item. It may be one of several types of integers, real or floating-point numbers, or ASCII character information. The data types are described in detail in the next section and summarized in tables 3-2 and 3-3.

DATA ENTRIES

A data entry is an ordered set of related data items. You specify the order of data items in an entry when you define the data base. Data entries may be defined with at most 127 data item names, none of which is repeated. The length of the data entry is the combined length of the data items it contains.

DATA SETS

A data set is.a collection of data entries where each entry contains values for the same data items. For example, the CUSTOMER data set contains entries composed of the same nine data items: ACCOUNT, LAST-NAME, FIRST-NAME, INITIAL, STREET-ADDRESS, CITY, STATE, ZIP, and CREDIT-RATING. Normally, each data set is associated with some real world entity such as orders, customers, employees, and so forth.

Each data set is referenced by a unique data set name. Each data set is stored in one disc file consisting of storage locations called records. When you describe the data base with the data base definition language, you specify the *capacity*, number of records, of each data set. Each record is identified by a record number which can be used to retrieve the entry within it.

DATA SET TYPES AND RELATIONS

An IMAGE data set is either a *master* or a *detail* data set. Figure 2-2 illustrates the relations between and types of six data sets in the STORE data base. Master data sets are identified by triangles and detail data sets by trapezoids. This convention is useful when diagramming the data base design.

MASTER DATA SETS

Master data sets are characterized in the following ways:

- They are used to keep information relating to a uniquely identifiable entity; for example, information describing a customer. The CUSTOMER data set in figure 2-3 illustrates this type of information.

- They allow for rapid retrieval of a data entry since one of the data items in the entry, called the *search item*, determines the location of the data entry. A search item may not be a compound item. In figure 2-3, the CUSTOMER data set contains a search item named ACCOUNT. The location of each entry is determined by the value of the customer's account number.

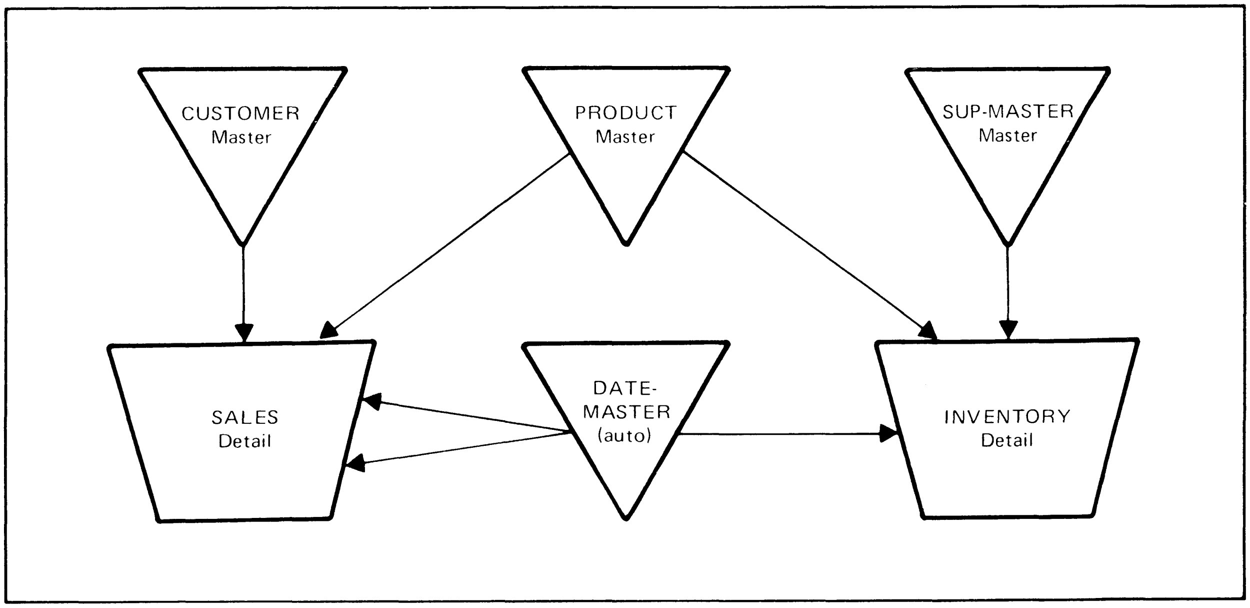


Figure 2-2. Master and Detail Data Set Relations

- They can be related to detail data sets containing similar search items and thus serve as indexes to the detail data set. The ACCOUNT search item in the CUSTOMER master data set is related to the ACCOUNT search item in the SALES detail data set. The entry for a customer named Abigail Brighton with account number 95430301 serves as an index to two entries in the SALES data set which contain information about purchases she made.

Although there are unused storage locations in the CUSTOMER data set, IMAGE disallows any attempt to add another data entry with account number 95430301. The search item value of each entry must remain unique. The values of other data items in the master data set are not necessarily unique. This is because they are not search items and are not used to determine the location of the data entry.

DETAIL DATA SETS

Detail data sets are characterized in the following ways:

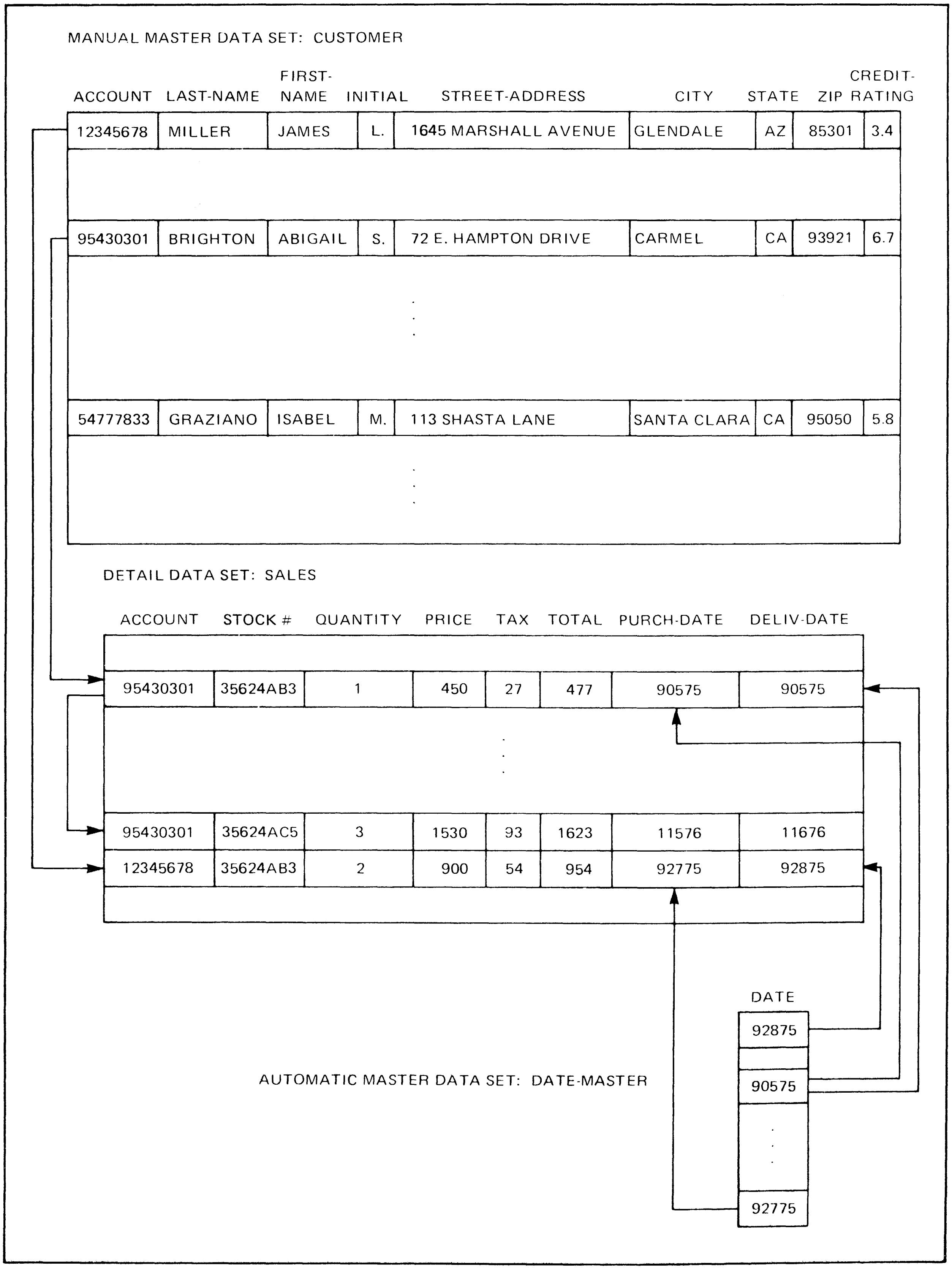
- They are used to record information about related events; for example, information about all sales to the same account.

- They allow retrieval of all entries pertaining to a uniquely identifiable entity. For example, account number 95430301 can be used to retrieve information about all sales made to Ms. Brighton.

The storage location for a detail data set entry has no relation to its data content. When a new data entry is added to a detail data set, it is placed in the first available location.

Unlike a master data set which contains at most one search item, a detail data set may be defined with from zero to 16 search items. The values of a particular search item need not be unique. Generally, a number of entries will contain the same value for a specific search item.

The SALES data set contains four search items: ACCOUNT, STOCK#, PURCH-DATE, and DELIV-DATE. Two entries in the example in figure 2-3 have identical values for the ACCOUNT item in the SALES data set.



MANUAL MASTER DATA SET: CUSTOMER

ACCOUNT	LAST-NAME	FIRST-NAME	INITIAL	STREET-ADDRESS	CITY	STATE	ZIP	CREDIT-RATING
12345678	MILLER	JAMES	L.	1645 MARSHALL AVENUE	GLENDALE	AZ	85301	3.4
95430301	BRIGHTON	ABIGAIL	S.	72 E. HAMPTON DRIVE	CARMEL	CA	93921	6.7
54777833	GRAZIANO	ISABEL	M.	113 SHASTA LANE	SANTA CLARA	CA	95050	5.8

DETAIL DATA SET: SALES

ACCOUNT	STOCK #	QUANTITY	PRICE	TAX	TOTAL	PURCH-DATE	DELIV-DATE
95430301	35624AB3	1	450	27	477	90575	90575
95430301	35624AC5	3	1530	93	1623	11576	11676
12345678	35624AB3	2	900	54	954	92775	92875

AUTOMATIC MASTER DATA SET: DATE-MASTER

DATE
92875
90575
92775

Figure 2-3. Master and Detail Data Sets Example

IMAGE stores pointer information with each detail data entry which links together all entries with the same search item value. Entries linked together in this way form a *chain.* A search item is defined for a detail data set if it is desired to retrieve together all entries with a common search item value, in other words, all entries in a chain. The SALES entries with ACCOUNT equal to 95430301 form a two-entry chain. A single chain may consist of at most 65535 entries.

PATHS

A master data set search item can be related to a detail data set search item of the same type and size. This relationship forms a *path.* A path contains a chain for each unique search item value. In figure 2-3, the ACCOUNT search item in CUSTOMER and the ACCOUNT search item in SALES link the CUSTOMER master to the SALES detail forming a path. One chain links all SALES entries for account number 95430301. The chain for account number 12345678 consists of one entry. Both chains belong to the same path.

Since a detail data set can contain as many as 16 search items, it can be related to at most 16 master data sets. Note that each master to detail relationship must be relative to a different detail search item. The SALES data set is related to the CUSTOMER, PRODUCT, and DATE–MASTER data sets.

A detail data set may be multiply indexed by a master data set. For example, SALES is indexed twice by DATE-MASTER. The DATE search item forms one path with the PURCH-DATE search item and one path with the DELIV-DATE search item.

Each master data set may serve as an index, singly or multiply, to one or more detail data sets. No master data set may be involved in more than 16 such relationships. For each such relationship, IMAGE keeps independent chain information with each master entry. This information consists of pointers to the first and last entries of the chain whose search item value matches the master set entry's search item value and a count of the number of entries in the chain. This is called a *chain head.* The format of chain heads is given in Section VII. For example, the DATE-MASTER data entries each contain two sets of pointers, one for PURCH-DATE chains and one for DELIV-DATE chains. Chain heads are maintained automatically by IMAGE.

AUTOMATIC AND MANUAL MASTERS

A master data set may be automatic or manual. These two types of masters have the following characteristics:

MANUAL	AUTOMATIC
May be 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.
You must explicitly add or delete all entries. A related detail data entry cannot be added until a master entry with matching search item value has been added. When the last detail entry related to a master entry 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.

MANUAL	AUTOMATIC
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 non stand-alone manual master can be used to prevent the entry of invalid data in the related detail data sets.	

EXAMPLE. In figure 2-3, CUSTOMER is a manual master data set and DATE-MASTER is an automatic master. Before the SALES entry for account 12345678 is added to SALES, CUSTOMER must contain an entry with the same account number. However, the DATE-MASTER entries for DATE equal to 92775 and 92875 are automatically added by IMAGE when the detail entry is added to SALES, unless they are already in the DATE-MASTER data set.

Note that DATE-MASTER contains only one data item, the search item DATE, while CUSTOMER, which is a manual master, contains several data items in addition to the search item.

If the SALES entry with account number 95430301 and stock number 35624AB3 are deleted and no other SALES entry contains a PURCH-DATE or DELIV-DATE value of 90575, the DATE-MASTER entry with that value is deleted automatically by IMAGE.

MANUAL VS. AUTOMATIC DATA SETS

Data base designers may use:

- manual masters to ensure that valid search item values are entered for related detail entries, or
- automatic masters to save time when the search item values are unpredictable or so numerous that manual addition and deletion of master entries is undesirable.

Whenever a single data item is sufficient for a master data set, the data base designer must decide between the control of data entry available through manual masters and the time-savings offered by automatic masters. For example, since DATE-MASTER is an automatic data set, erroneous dates such as 331299 may be entered accidentally.

PRIMARY PATHS

One of the paths of each detail data set may be designated by the data base designer as the *primary path.* The main reason for designating a path primary is to maintain the entries of each chain of the path in contiguous storage locations. You accomplish this by occasionally using the DBUNLOAD utility program to copy the data base to tape, the DBUTIL utility program to erase the data base, and the DBLOAD program to reload the data base from the tape. When the data base is reloaded, contiguous storage locations are assigned to entries of each primary path chain. Therefore, the data base designer should designate the path most frequently accessed in chained order as the primary path. This type of access is discussed in Section IV.

A primary path also serves as the default path when accessing a detail data set if no path is specified by the calling program. This characteristic of primary paths is described with the DBGET procedure in Section IV.

SORT ITEMS

For any path, it is possible to designate some data item other than the search item as a *sort item.* If a sort item is specified, each of the chains of the path are maintained in ascending sorted order, based on the values of the sort item. Different paths may have different sort items, and one path's sort item may be another path's search item. Only data items of type logical or character can be designated as sort items.

For example, chains in the SALES data set composed of entires with identical ACCOUNT values are maintained in sorted order by PURCH-DATE. When information about sales to a particular customer is required, the SALES data entries for that customer's account can be retrieved in sorted order according to purchase date. (For PURCH-DATE to be a meaningful sort item, dates must be stored in a properly collatable form such as year-month-day rather than the order shown in preceding figures.)

The sorted order of entries is maintained by logical pointers rather than physical placement of entries in consecutive records. Figure 2-4 illustrates the way in which sorted paths are maintained by IMAGE. When an entry is added to a detail data set it is added to or inserted in a chain. If the path does not have a sort item defined, the entry follows all existing entries in the chain. If the path has a sort item, the entry is inserted in the chain according to the value of that item.

If the entry's sort item value matches the sort item values of other entries in the chain, the position of the entry is determined by an extended sort field consisting of the sort item value and the values of all items following the sort item in the entry. If the extended sort field matches another extended sort field, the entry is inserted chronologically following the other entries with the same extended sort field value. This also occurs if the sort item is the last item in the entry and its value matches another entry's sort item value.

When the data base content is copied to magnetic tape using the IMAGE utility program DBUNLOAD, the pointers that define an entry's position in a chain are not copied to the tape. When the data is loaded back into the data base, the chains are recreated. Therefore, entries which were previously ordered chronologically will not necessarily be in that same order. The new chronological ordering is based on the order in which the entries are read from the tape. The chains of a primary path are an exception; the order of these chains is preserved if the tape was created with DBUNLOAD in the chained mode. (Section VI contains more information about DBUNLOAD.)

NOTE

It is important to limit the use of sorted chains to paths consisting of relatively short chains. It is not intended that sorted paths be used for multiple key sorts, or for sorting entire data sets. These functions are handled more efficiently by user-written routines or the MPE subsystem, Sort/3000.

THE STORE DATA BASE

Figures 2-5 and 2-6 illustrate the complete STORE data base. Figure 2-5 lists the data items that define entries in each data set. The data type is in parentheses. (Data types are described in Section III with the item part of the schema.) Paths are indicated by arrows. CUSTOMER, SUP-MASTER, PRODUCT, and DATE-MASTER are master data sets and SALES and INVENTORY are detail sets. Figure 2-6 shows a sample entry from each data set except DATE-MASTER for which it shows two sample entries.

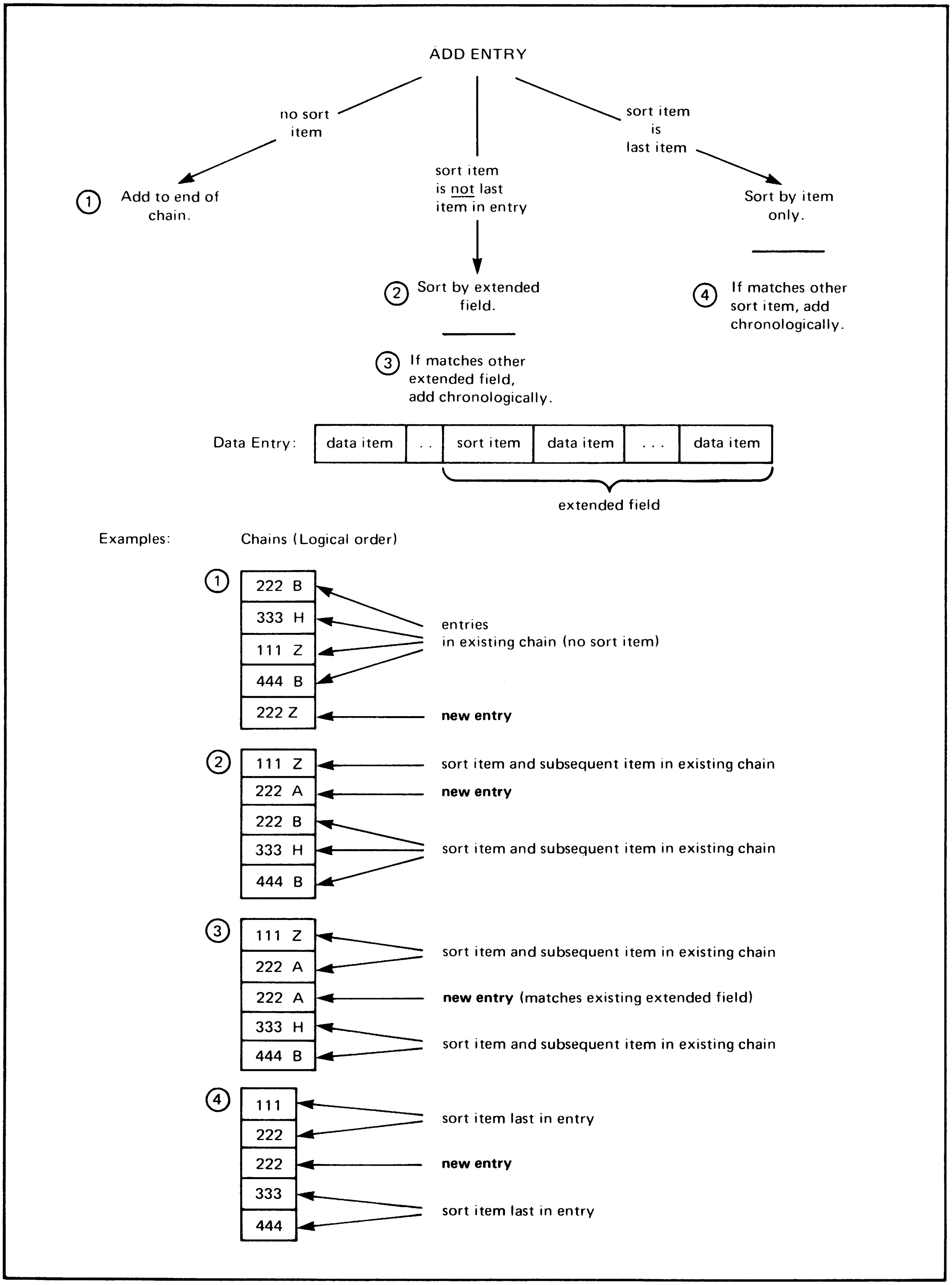

ADD ENTRY
no sort item
sort item is not last item in entry
sort item is last item
① Add to end of chain.
Sort by item only.
② Sort by extended field.
④ If matches other sort item, add chronologically.
③ If matches other extended field, add chronologically.
Data Entry:
data item
. .
sort item
data item
. . .
data item
extended field
Examples:
Chains (Logical order)
①
222 B
333 H
111 Z
444 B
222 Z
entries in existing chain (no sort item)
new entry
②
111 Z
222 A
222 B
333 H
444 B
sort item and subsequent item in existing chain
new entry
sort item and subsequent item in existing chain
③
111 Z
222 A
222 A
333 H
444 B
sort item and subsequent item in existing chain
new entry (matches existing extended field)
sort item and subsequent item in existing chain
④
111
222
222
333
444
sort item last in entry
new entry
sort item last in entry

Figure 2-4. Adding an Entry to a Sorted Chain

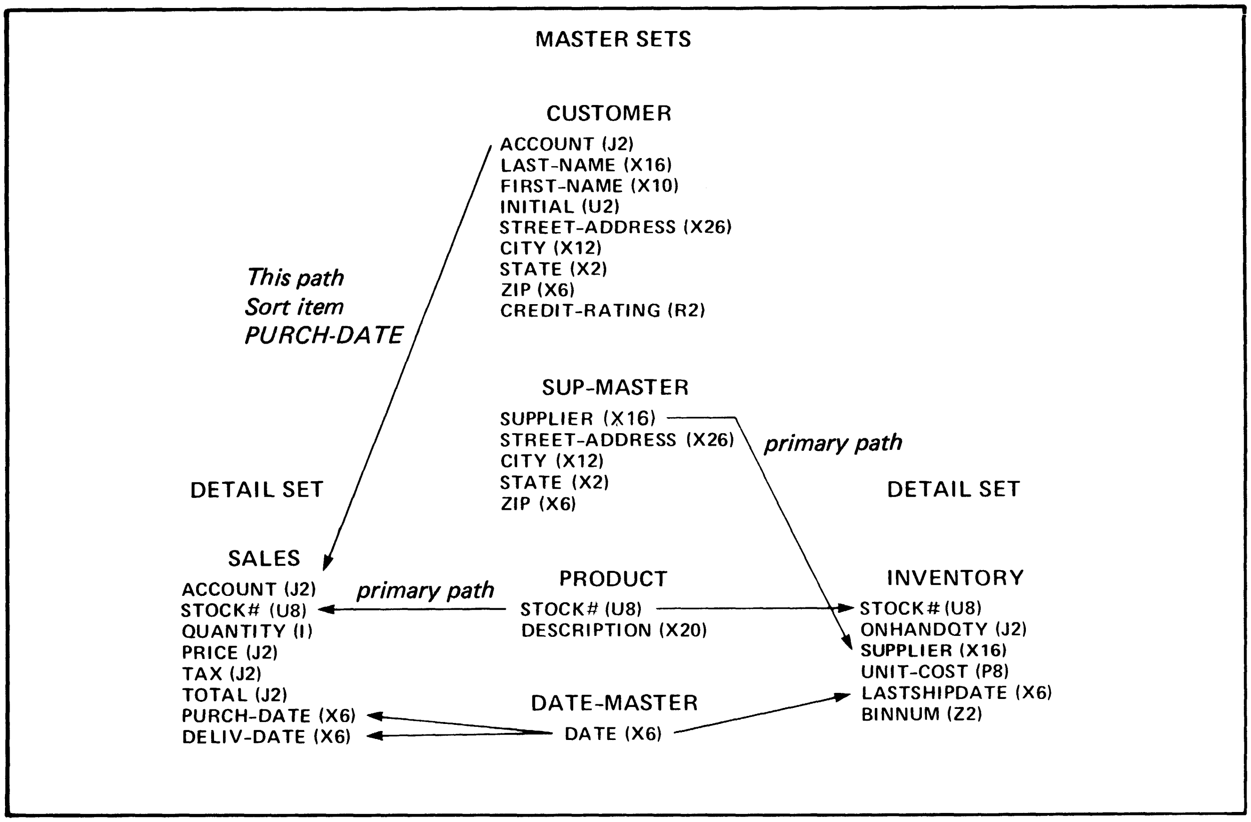


Figure 2-5. STORE Data Sets and Paths

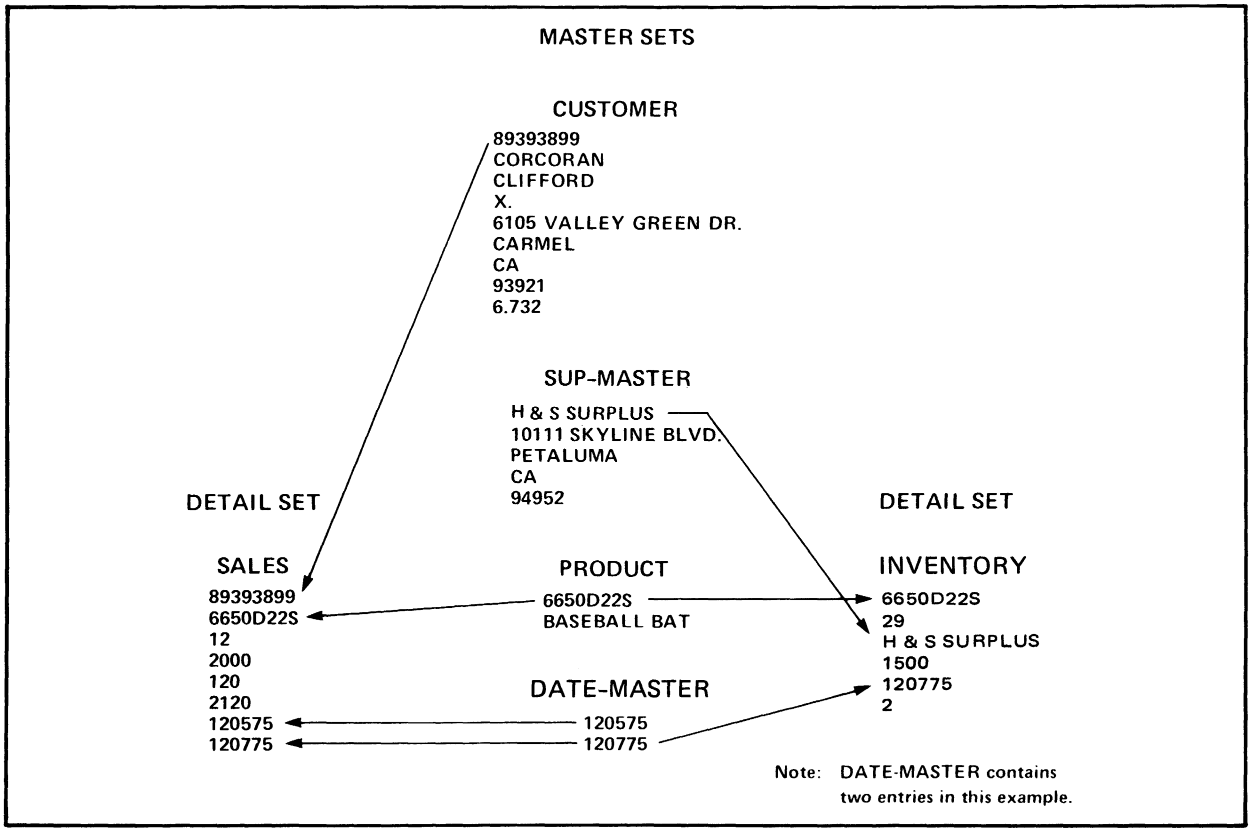


Figure 2-6. Sample Entries for STORE Data Sets

Chains of the path formed by CUSTOMER and SALES are maintained in sorted order according to the value of PURCH-DATE. The primary path for INVENTORY is the one defined by SUP-MASTER and the primary path for SALES is the one defined by PRODUCT.

DATA BASE FILES

Data base elements are stored in privileged MPE disc files. In addition to the root file which contains the data base definition, other files called *data files* contain the data sets.

ROOT FILE

The root file is created for the data base creator when he or she executes the Schema Processor. It is catalogued within the creator's log-on group and account with a local file name identical to the data base name. Thus, the name of the root file for the STORE data base is STORE. Refer to the *MPE Commands Reference Manual* for more information about MPE accounts and log-on groups.

The root file is a single-extent MPE disc file: that is, the entire file occupies contiguous sectors on the disc. It serves as a common point of entry to and source of information about the data base.

DATA FILES

There is one data file for each data set of a data base. The size of each record and number of records in the file are determined by the contents of the root file. The data files are created and initialized with the IMAGE utility program, DBUTIL.

Each data file is catalogued within the same group and account as the root file. Local file names are created by appending two digits to the local name of the root file. These two digits are assigned to the data sets according to the order in which they are defined in the schema. For example, the STORE data base is defined with CUSTOMER and DATE-MASTER as the first two data sets. These data sets are in data files STORE01 and STORE02.

Each data file is physically constructed from one to 32 extents of contiguous disc sectors, as needed to meet the capacity requirements of the file, subject to the constraints of the MPE file system. Each data file contains a user label in a disc sector maintained and used by the IMAGE library procedures. The label contains structural pointers and counters needed for dynamic storage allocation and deallocation.

RECORD SIZE. Record sizes vary between data files but are constant within each file. Each record is large enough to contain a data entry and the associated IMAGE pointer information. The amount of pointer information depends on the way the data set is defined. Pointer information is described in Section VII. The maximum number of records in a data set file depends on the record size, the available disc space, and the MPE file system constraints.

BLOCKS. The records in a data file are physically transferred to and from the disc in groups. Each group involved in a single disc transfer is called a *block*. The number of records in each block is called the *blocking factor*. The Schema Processor determines the blocking factor during creation of the root file. Section III contains more information about block size and blocking factors in the discussion of the set part of the schema. The format of blocks is given in Section VII.

PROTECTION OF THE DATA BASE

IMAGE prevents unauthorized persons from gaining access to the data base. It provides external protection through the MPE privileged file, account, and group structures and, in addition, provides the data base designer and data base manager with devices for further protection of the data base.

PRIVILEGED FILE PROTECTION

All IMAGE data base files are privileged files. (See the *MPE Intrinsics Reference Manual* for a description of the MPE privileged file capability.) Access by unprivileged processes or through most MPE file system commands is not allowed. Therefore, non-privileged users are prevented from accidentally or deliberately gaining access to the data base.

The use of MPE commands that permit copying of IMAGE files to tape, represent a potential breach of data base privacy, and their use should be controlled. In particular, anyone who uses the SYSDUMP, STORE, or RESTORE commands should notify the data base manager. The SYSDUMP and STORE commands permit system supervisors, system managers and other privileged users to copy files not currently open for output to tape. The MPE RESTORE command may purge and replace a data base file with a different file if it has the same name and is encountered on tape.

ACCOUNT AND GROUP PROTECTION

In order to gain access to an IMAGE data base, you must be able to access the files in the account and group in which the data base resides. The system manager and account manager manage the security levels for accounts and groups. The system manager is responsible for creating accounts and the account manager for creating new groups and users. (The *System Manager/System Supervisor Reference Manual* contains detailed information about the maintenance of MPE accounts and groups.)

The system and account managers can prevent members of other accounts from accessing the data base by specifying user type AC (Account Member) for the account and group containing the data base. They can prevent users who are members of the account, but not of the group, containing the data base from accessing it by specifying GU (Group User) for the group. On the other hand, they can allow access from other accounts by specifying user type ANY at both the account and group levels.

These MPE security provisions provide an account and group level of security controlled by the system manager and account manager.

USER CLASSES AND PASSWORDS

IMAGE allows the data base designer to control access to specific data sets and data items by defining up to 63 *user classes* and then associating the user classes with data sets and data items in *read* or *write class lists.* This association determines which user classes may access which data elements and the type of access that is granted.

Each user class is identified by an integer from 1 to 63 and is associated with a *password* defined by the data base designer. For example, the STORE data base is defined with these user classes and passwords:

User Class	Password
11	CREDIT
12	BUYER
13	SHIP-REC
14	CLERK
18	DO-ALL

The magnitude of the user class number has no relation to the capability it grants.

When you initiate access to the data base, you must supply a password to establish your user class. If the password is null or does not match any password defined for the data base, the user class assigned is zero. This does not apply if you are the data base creator and supply a semicolon in which case you have full access to all data sets in the data base. IMAGE uses the number 64 to identify the data base creator.

READ CLASS LISTS AND WRITE CLASS LISTS. When the data items and data sets are defined in the schema, a read class list and a write class list can be specified for each item or set. Table 2-1 contains sample lists for the CUSTOMER data set and CREDIT-RATING data item in the STORE data base.

Table 2-1. Sample Read/Write Class Lists

	READ CLASS LIST	WRITE CLASS LIST
CUSTOMER	11, 14	11, 18
CREDIT-RATING	14	14

User class numbers included in the write class list are, by implication, included in the read class list. Since a write class list of 14 implies that user class 14 is in the read class list, the CREDIT-RATING read class list is redundant. However, it may be included as a reminder in the schema of the total capability granted to user class 14.

A distinction must be made between the absence of a read and write class list and a null list. When you specify the lists in the schema, they are enclosed in parentheses and separated by a slash, for example, (11, 14/15). A null list may be one of the following:

(/) Both read and write class lists are null.

(11, 14) The write class list is null.

Since the existence of a write class list implies a read class list, there is no situation where only the read class list is null.

The absence of both a read and write class list, and the parentheses and slash, yields the same result as a read class list containing all user classes and write class list which is null. For example:

(0,1,2,3, . . . 63 /)

The effect of null and absent lists is illustrated later in this section.

ACCESS MODES AND DATA SET WRITE LISTS

Before you can gain access to a data base, you must open it specifying a password that establishes your user class number and an *access mode* that defines the type of data base tasks you want to perform. Access modes are described in Section IV with the instructions for opening a data base. At this time it is necessary only to note that some of the eight available access modes nullify the data set write list. If the data base is opened in access mode 2, 5, 6, 7, or 8, all data set write class lists are effectively null and the user class numbers in the write class lists are in the data set read class lists only. This effect should be considered when you are designing the security scheme for the data base.

GRANTING A USER CLASS ACCESS TO A DATA ELEMENT

Tables 2-2 and 2-3 illustrate the use of read and write class lists from two different perspectives. Table 2-2 shows what capability user class 11 has if it appears in the lists as shown. The same rules apply to any user class. The access mode must be as indicated.

A null read and write class list can be used by the data base creator at the data set level to deny access to the data set by all user classes; that is, only the data base creator will be able to use the data set.

Table 2-2. Granting Capability to User Class 11

	LIST	CAPABILITY	LIST	CAPABILITY	LIST	CAPABILITY
Control at Data Set Level	(/11) or (11/11)	Total access to set if *access mode* 1, 3, or 4	(/)	No access to set	(11) or absent list	Controlled at item level
Control at Data Item Level	(/11) or (11/11)	Update and read item	(/)	No access to item	(11) or absent list	Read item

Table 2-3 presents the same rules organized by the task which the user class is to perform. It lists the required access modes and the security rules at both the data set and data item level. For simplicity assume there are always read and write class lists even if they are the default lists (0, 1, 2, . . . 63/) resulting when the lists are not actually specified in the schema (absent lists).

In summary, the data base designer can grant access to a data set in the following ways:

- **Specify the user class number in the data set write class list.**

  If the data base is opened in access mode 1, 3, or 4, this grants the user class complete access to the data set. Users in this class can add and delete entries, update the value of any data item that is not a search or sort item, and read any item, regardless of the data item read and write class lists. A user class number must be in the data set write list in order to add and delete entries.

Table 2-3. Enabling a User Class to Perform a Task

TASK	READ DATA ITEM	UPDATE DATA ITEM	ADD OR DELETE DATA ENTRIES
Access Modes	1 – 8	1 – 4	1, 3, 4
Data Set Security Rules	If access mode 1, 3, 4: User class in write list OR User class in read list and pass data item security. If access mode 1, 5-8: User class in read or write list and pass data item security.	If access mode 1, 3, 4: User class in write list OR User class in read list and pass data item security. If access mode 2: User class in read or write list and pass data item security.	User class in data set write list.
Data Item Security Rules	User class in read or write list.	User class in write list.	

Note: There are other considerations in selecting the access mode. These are discussed in Section IV.

If the data base is opened in access mode 2, 5, 6, 7, or 8, this is the same as specifying the user class number in the data set read class list only and the next rule applies.

- **Specify the user class number in the data set read class list** (or omit both lists entirely).

  This grants the user class a type of access to the data set that is controlled at the data item level as described below. If both read and write class lists are omitted, the user class is granted this type of access since the lists are (0, 1, 2, ... 63/) by default.

- **Omit the user class number from both the specified read and write class lists.**

  This denies the user class any type of access to the data set.

Assuming the data base designer has established control at the data set level as summarized above, control at the data item level is established in the following ways:

- **Specify the user class number in the data item read class list** (or omit both lists entirely).

  This grants the user class read access to the data item.

- **Specify the user class number in the data item write class list.**

  This grants the user class the ability to update or change the data item value, if it is not a search or sort item. Since the user class is implied to be in the read class list, the user class can also read the item. A user class number must be in the data item write list in order to change the value.

- **Omit the user class number from both the read and write class list.**

  This denies the user class any type of access to the data item.

The protection of data set and data item values is designed so that the data base designer must explicitly specify the user class number to allow that class to make any type of change to the data base, but read access may be granted by default in some situations, for example, by omitting the lists entirely. To deny read access to a data set or data item, the data base designer must specify a list, possibly a null one, and deliberately omit the user class number.

EXAMPLES. In the STORE data base, only user classes 11 and 18 can add and delete CUSTOMER data entries since these are the only user class numbers in the data set write list as shown in table 2-1. To do so, they must open the data base in access mode 1, 3, or 4.

User class 14 can update the CREDIT-RATING data item in the CUSTOMER data set because it is in the data item write list and the data set read list.

Table 2-4 contains more illustrations of the effects of read and write class lists. The data base creator and user class 9 (in access mode 1, 3, or 4) have complete access to data set 1 but only the creator has complete access to data set 2. Complete access includes the ability to read and update all items and add and delete entries.

Table 2-4. Sample Read and Write Class Lists

Data Set A	(0,18,13/9)	**Item Read Access**	**Item Update Access**
Data Item A		0, 13, 18, 9	9*
Data Item B	(/13)	13, 9*	13, 9*
Data Item C	(/)	9*	9*
Data Item D	(/9)	9	9
Data Item E	(18/13)	13, 18, 9*	13, 9*
Data Item F	(/13, 18)	13, 18, 9*	13, 18, 9*
Data Item G	(12/0)	0, 9*	0, 9*
Data Item H	(13/)	13, 9*	9*
Data Set B			
Data Item A		0, 1, . . . , 63	
Data Item I	(13/9)	13, 9	9

*Only if access mode is 1, 3, or 4.
None of these items are search or sort items.

USER CLASSES AND LOCKING

IMAGE does not consider user classes when locking a data base entity. Any data set or any data item can be referenced in a lock request by any user of a data base regardless of his or her user class.

PROTECTION IN RELATION TO LIBRARY PROCEDURES

All access to a data base is achieved through a Data Base Control Block (DBCB) and one or more User Local Control Blocks (ULCBs) which reside in privileged data segments not directly accessible to data base users. Since no user process can read or modify these control blocks, IMAGE guarantees protection of the data base from unauthorized programmatic access. See the description of the DBCB and ULCB in Section VII. For more information about data segments and privileged mode, see the *MPE Intrinsics Reference Manual.*

All IMAGE library procedures that structurally modify the data base execute in "critical mode." This defers any requested process termination while modifications are in progress. If any file system failures occur during such data base modification, IMAGE causes process termination since the data base integrity is suspect.

The DBCB contains buffers which are used to transfer data. All buffers whose content has been changed to reflect a modification of the data base are always written to disc before the library procedure exits to the calling program. This guarantees data integrity despite any program termination that might occur between successive procedure calls.

PROTECTION PROVIDED BY THE IMAGE UTILITIES

The IMAGE utilities perform various checks to ensure data base integrity.

- They acquire exclusive or semi-exclusive access to the data base being processed. (Section IV contains more information about types of access in the discussion of opening a data base.)
- Only the data base creator or a user supplying the correct *maintenance word* can execute the utilities. The data base creator defines the maintenance word when the data base is created with the DBUTIL utility program. (Refer to Section VI.) Anyone running the utility programs must be logged on to the group in which the data base is catalogued.
- Unrecoverable disc or tape problems are treated as functional failures rather than limited successes and result in program termination.

DEFINING A DATA BASE

SECTION III

Once the data base has been designed, it must be described with the data base description language and processed by the Schema Processor to create the root file: Figure 3-1 illustrates the steps in defining the data base.

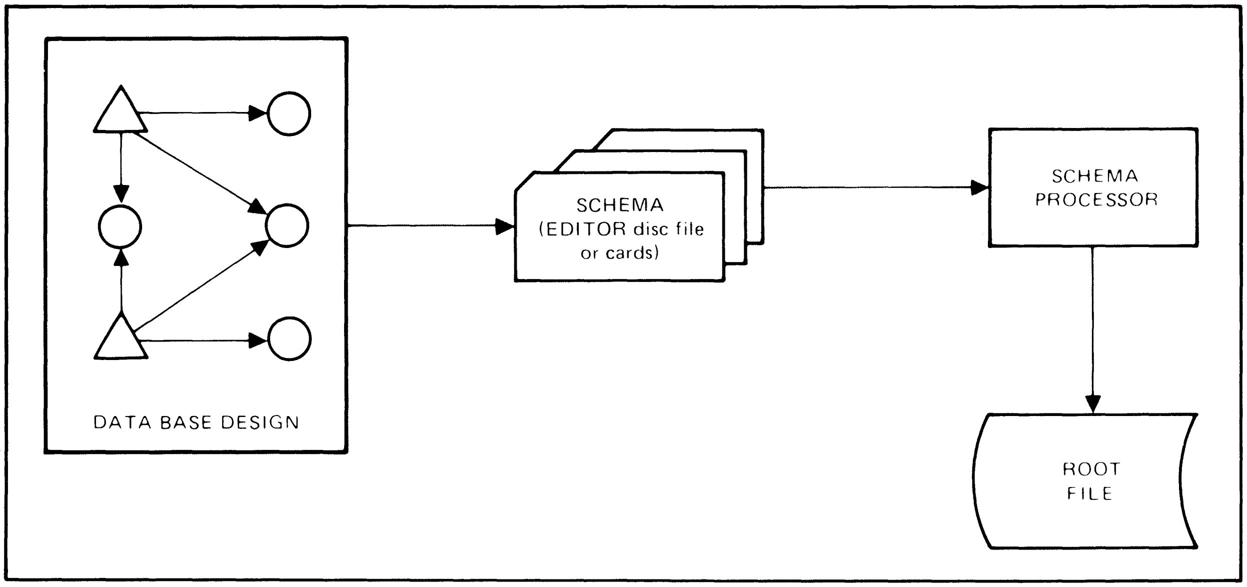


Figure 3-1. Data Base Definition Process

DATA BASE DESCRIPTION LANGUAGE

The data base description, called a schema, may exist in the MPE system as an ASCII file on cards, magnetic tape, or as a catalogued disc file. Regardless of the actual physical record size of the file, the Schema Processor reads, prints, and processes only the first 72 characters of each record. Any remaining character positions in the record are available for your convenience, to be used for comments or collating information. The data base description language is a free-format language; you can insert blanks anywhere in the schema to improve its appearance except within symbolic names and reserved words.

LANGUAGE CONVENTIONS

The conventions used in describing the data base language are the same as those described on the conventions sheet at the beginning of this manual. In addition, the conventions in table 3-1 apply.

Table 3-1. Additional Conventions

Punctuation	All punctuation appearing in format statements must appear exactly as shown.
Comments	Comments take the form: <<*comment*>> They may contain any characters and may appear anywhere in the schema except embedded in another comment. They are included in the schema listing but are otherwise ignored by the Schema Processor program.
Data Names	Data names may consist of from 1 to 16 alphanumeric characters, the first of which must be alphabetic. Characters after the first must be chosen from the set: letters A – Z, digits 0 – 9, or + - * / ? ' # % & @
Upshifting	All alphabetic input to the Schema Processor is upshifted (converted to upper case), with the exception of passwords which may contain lowercase characters.

SCHEMA STRUCTURE

The overall schema structure is:

BEGIN DATA BASE *data base name;*
PASSWORDS: *password part*
ITEMS: *item part*
SETS: *set part*
END.

The *data base name* is an alphanumeric string from 1 to 6 characters. The first character must be alphabetic.

The *password part*, *item part*, and *set part* are described on the following pages. Figure 3-5 contains a complete schema for the STORE data base that is used in the examples in this manual.

PASSWORD PART

The password part defines user classes and passwords. Section II contains a description of user classes and how they are used to protect data elements from unauthorized access.

> The form of the password part is
>
> *user class number* [*password*] ;
>
> .
>
> .
>
> .
>
> *user class number* [*password*] ;

For example,

```
5   GLOWWORM;
61  REDHOT;                      password
12  DOZEN;
  user class number
```

where

user class number is an integer between 1 and 63 inclusive. User class numbers must be unique within the password part.

password may consist of from 1 to 8 ASCII characters including lower case and excluding carriage return, semicolon, and blank. Blanks are removed by the Schema Processor.

If the same password is assigned to multiple user class numbers, the highest numbered class is used. It is not an error to omit the *password*, but the Schema Processor ignores lines containing only a user class number.

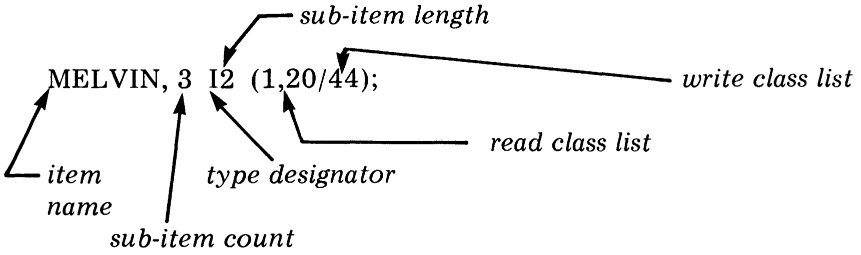
ITEM PART

The item part defines data items including the data item name, length, and the user classes that have access to the item. The data set in which the data item appears is defined in the set part definition.

The form of the item part is

> *item name*, [*sub-item count*] *type designator* [*sub-item length*]
> [(*read class list*/*write class list*)];

For example,



```
MELVIN, 3 I2 (1,20/44);
```

where

item name is the data item name. It must be a valid IMAGE data name as described in table 3-1. It must be unique within the item part.

sub-item count is an integer from 1 to 255 that denotes the number of sub-items within an item. If omitted, by default it equals one. A data item whose sub-item count is 1 is a simple item. If the sub-item count is greater than one, it is a compound item.

type designator defines the form in which a sub-item value is represented in the computer. The type designators I, J, K, R, U, X, Z, P are described in table 3-2.

sub-item length is an integer from 1 to 255. It is the number of words, characters, or nibbles (depending on the type designator) in a sub-item. If omitted, it is equal to 1 by default.

read class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in Section II.

write class list is a group of user class numbers between 0 and 63, inclusive, separated by commas.

There can be no more than 255 data items in a data base. A data item name can appear in more than one data set definition. For example, a data item named ACCOUNT appears in both the CUSTOMER and SALES data sets of the STORE data base.

DATA ITEM LENGTH

Each data item value is allotted a storage location whose length is equal to the product of the item's *sub-item length* and its *sub-item count.* The unit of measure for the length depends upon the type designator and may be a word, byte, or nibble. A word is a 16-bit computer word, a byte is eight bits or a half-word, and a nibble is four bits or a half-byte. Table 3-2 defines the various type designators and specifies the unit of measure used for each.

Table 3-2. Type Designators

WORD DESIGNATORS	
I	A signed binary integer in 2's complement form.
J	Same as I but QUERY allows only numbers conforming to specifications for COBOL COMPUTATIONAL data to be entered.
K	An absolute binary quantity.
R	A real (floating point) number.
CHARACTER DESIGNATORS	
U	An ASCII character string containing no lowercase alphabetic characters.
X	An unrestricted ASCII character string.
Z	A zoned decimal format number.
NIBBLE DESIGNATOR	
P	A packed decimal number.

A data item must be an integral number of words in length regardless of the type designator and its unit of measure. In other words, data items of type U, X, or Z which are measured in bytes must have a sub-item length and sub-item count such that their product is an even number. If a data item is defined as U3, it cannot be a simple item and must have an even numbered sub-item count so that the data item length is an integral number of words. Data items of type P which are measured in nibbles must have a sub-item length and sub-item count such that their product is evenly divisible by 4, since 4 nibbles equal 1 word.

A data item cannot exceed 2047 words in length. The entire item, whether simple or complex, is always handled as a unit by IMAGE.

IMAGE DATA TYPES AND PROGRAM LANGUAGE DATA TYPES

The *type designator*, *sub-item count*, and *sub-item length* you specify for a data item defines its length. IMAGE does not perform any conversions of data or examine the item to check its validity as it is being added to the data base. The only data item values that IMAGE checks are those specified as part of a lock descriptor in calls to the DBLOCK procedure. (Refer to the discussions of locking in Section IV.) There are no rules that a specific type of data defined by a programming language must be stored in a specific type of IMAGE data item.

Table 3-3 relates IMAGE type designators and sub-item lengths to the data types typically used to process them in the available programming languages. Some BASIC language restrictions are noted.

ITEM PART

Table 3-3. IMAGE Type Designators and Programming Languages

	COBOL	FORTRAN	RPG	SPL	BASIC
I	COMPUTATIONAL S9 to S9(4)	INTEGER	Binary	INTEGER	INTEGER**
I2	COMPUTATIONAL S9(5) to S9(9)	INTEGER*4	Binary	DOUBLE INTEGER	
I4	COMPUTATIONAL S9(10) to S9(18)		Binary		
J	COMPUTATIONAL S9 to S9(4)	INTEGER	Binary	INTEGER	INTEGER**
J2	COMPUTATIONAL S9(5) to S9(9)	INTEGER*4	Binary	DOUBLE INTEGER	
J4	COMPUTATIONAL S9(10) to S9(18)		Binary		
K1		LOGICAL		LOGICAL	***
R2*		REAL		REAL	REAL****
R4		DOUBLE PRECISION		LONG	LONG****
U	DISPLAY PICTURE A	CHARACTER	Character	BYTE	String
X	DISPLAY PICTURE X	CHARACTER	Character	BYTE	String
Z	DISPLAY PICTURE 9		Character		
P	COMPUTATIONAL-3		Numeric		

*Real numbers must have a length of 2 or more words; R and R1 cannot be used by IMAGE.

**BASIC integers cannot have the value −32768.

***Type LOGICAL items >32767 which are accessed as type INTEGER in BASIC programs are treated as negative integers.

****BASIC REAL and LONG data cannot have the value 10^{-78}

Note that the UNIT-COST item in the INVENTORY data set is easier to process with COBOL or RPG programs than with the other languages since packed data is a standard data type in COBOL and RPG. However, the CREDIT-RATING data item in the CUSTOMER data set is easier to process with FORTRAN, SPL, or BASIC programs since real numbers can be arithmetically manipulated in these languages. An actual data base may be designed so that some data sets are processed by programs coded in one language and others by programs coded in another language. Another data set may be conveniently processed by programs written in any of the languages.

DATA ITEMS OF TYPE P

The bits used to represent the sign of a packed decimal value may vary depending on whether the value is entered using QUERY, a COBOL program, or an RPG program. Here is a summary of what happens in each case:

- If a value is entered using QUERY, and no sign is specified, the sign is 1111_2.
  If a value is entered using QUERY, and a plus sign is specified, the sign is 1100_2.
  If a value is entered using QUERY, and a minus sign is specified, the sign is 1101_2.
- If a value is entered using a COBOL program, and the PICTURE clause does not specify a sign, the sign is 1111_2.
  If the PICTURE clause specifies a sign and the value is positive, the sign is 1100_2.
  If the PICTURE clause specifies a sign and the value is negative, the sign is 1101_2.
- If a value is entered using an RPG program, a positive or unsigned value's sign is 1100_2 and a negative value's sign is 1101_2.

When you use IMAGE to locate all packed data items with a particular value (as described later in this manual), you must be aware that IMAGE differentiates between unsigned, positive, and negative data items with the same absolute value. For example, if you search for all data items with the value +2, IMAGE will not retrieve any items with the unsigned value 2.

In general, IMAGE treats any two values with different binary representations as unequal regardless of their type.

COMPLEX NUMBERS. Applications programmed in BASIC or FORTRAN can define and manipulate complex numbers by using data type R2 with a sub-item count of 2, storing the real part in the first sub-item and the imaginary part in the second sub-item.

QUERY AND DATA TYPES. QUERY supports only a subset of the available data item types. If you intend to use QUERY you should consult the *QUERY Reference Manual* for specific information about the way QUERY handles the various IMAGE data types, including compound data items.

Table 3-4. Examples of an Item Part

```
ITEMS:

A,I2;                    << 32 BIT SIGNED INTEGER>>

MELVIN,3I (1,20/44);     << COMPOUND ITEM. THREE SINGLE WORD SIGNED INTEGERS.
                            READ CLASSES ARE 1 AND 20; WRITE CLASS IS 44*>>

BLEVET,J;                << SINGLE-WORD SIGNED INTEGER BETWEEN -9999 AND 9999.>>

COSTS, 2X10;             << COMPOUND ITEM. TWO 10-CHARACTER ASCII STRINGS.>>

DATE, X6;                << SIX-CHARACTER ASCII STRING.>>

VALUES, 20R2(1/8);       << COMPOUND ITEM. 20 2-WORD REAL (FLOATING-POINT) NUM-
                            BERS. READ CLASS IS 1; WRITE CLASS IS 8*>>

PURCHASE-MONTH, U8;      << EIGHT-CHARACTER ASCII STRING WITH NO LOWER CASE
                            ALPHABETICS.>>

MASK, K2;                << 32 BIT ABSOLUTE BINARY QUANTITY.>>

TEMPERATURE, 17R4;       << COMPOUND ITEM. 17 FOUR WORD REAL (FLOATING-POINT)
                            NUMBERS.>>

SNOW*#@,Z4;              << FOUR-DIGIT ZONED DECIMAL (NUMERIC DISPLAY) NUMBER.>>

POPULATION,P12;          << 11 DECIMAL DIGITS PLUS A SIGN IN THE LOW ORDER NIBBLE.
                            OCCUPIES THREE WORDS.>>
*WRITE CLASSES CAN ALSO READ.
```

DATA ITEM IDENTIFIERS

When you use the IMAGE procedures described in the next section, you can reference a data item by name or number. The data item number is determined by the item's position in the item part of the schema. The first item defined is item one, the second is item 2, and so forth.

It is more flexible to use data item names since a change in the order of the item definitions or the deletion of an item definition from the schema might require changes to all application programs referencing the data items by number. Thus, to maintain program file independence it is recommended that you use data item names if possible.

SET PART(MASTERS)

The set part of the schema defines data sets. It indicates which data items listed in the item part belong to which sets and links the master data sets to the detail data sets by specifying search items.

The form of the set part for Master Data Sets is

$$\left\{\begin{array}{l}\text{NAME:}\\ \text{N:}\end{array}\right\}\quad \textit{set name,}\quad \left\{\begin{array}{l}\text{MANUAL}\\ \text{M}\\ \text{AUTOMATIC}\\ \text{A}\end{array}\right\}\quad [(\textit{read class list/write class list})];$$

$$\left\{\begin{array}{l}\text{ENTRY:}\\ \text{E:}\end{array}\right\}\quad \textit{item name}\quad [(\textit{path count})],$$

$$\begin{array}{l}\cdot\\ \cdot\end{array}$$

$$\textit{item name}\quad [(\textit{path count})];$$

$$\left\{\begin{array}{l}\text{CAPACITY:}\\ \text{C:}\end{array}\right\}\quad \textit{maximum entry count;}$$

For example,

```
      set name ──╲      read class list ──╲  ╱── write class list
                  ↘                        ↘↙
NAME:          CUSTOMER,MANUAL (1,4/1,8);
                  ↙──────────────────────────── item name
ENTRY:         ACCOUNT(1),
                         ↙───────────────────── path count
               LAST-NAME,
               FIRST-NAME,
               INITIAL,
               STREET-ADDRESS,
               CITY,
               STATE,
               ZIP,
               CREDIT-RATING;
CAPACITY:20003; ←─────────────────── maximum entry count
```

where

set name is the data set name. It must be a valid IMAGE data name as described in table 3-1.

MANUAL (or M) denotes a manual master data set. Each entry within a manual master must be created manually and may contain one or more data items.

AUTOMATIC (or A) denotes an automatic master data set. Each data entry within an automatic master is created automatically by IMAGE and contains only one data item.

read class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in Section II.

write class list is a group of user class numbers between 0 and 63, inclusive, separated by commas.

item name is the name of a data item defined in the item part. A search item defined by a path count must be a simple item.

path count is an integer between 0 and 16, inclusive, which is used with the search item only. It indicates the number of paths which will be established to various detail data sets. (See Section II for more information about paths.) A path count must be specified for one, and only one, item in the master set. A zero path count may be used with a manual master data item to indicate the search item. A manual master defined in this way is not linked to any detail data set. An automatic master has one item that must have a path count greater than zero.

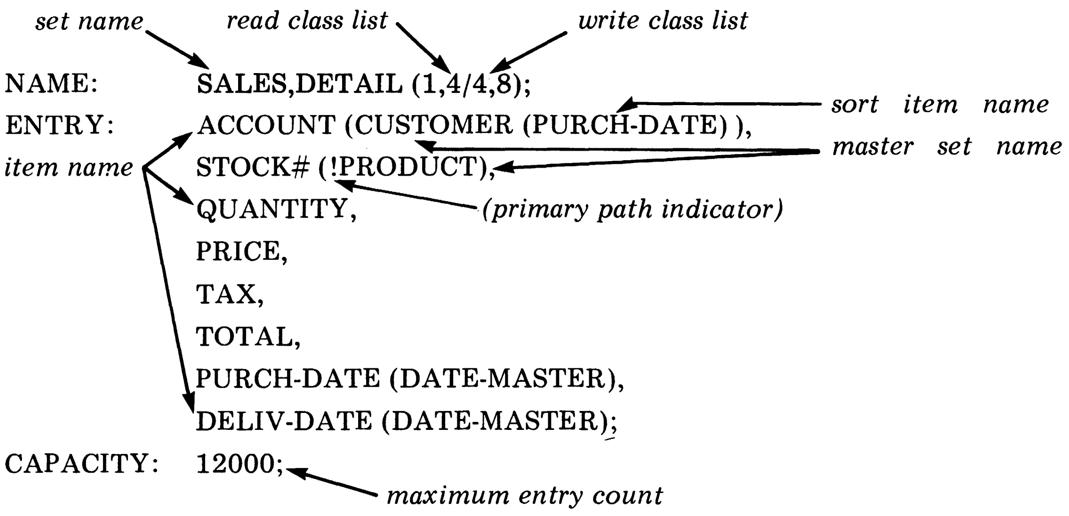
maximum entry count is the maximum number of entries the data set can contain, the data set's capacity. It must be less than 2^{23} (8,388,608).

SET PART(DETAILS)

The form of the set part for Detail Data Sets is

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

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

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

For example



```
NAME:        SALES,DETAIL (1,4/4,8);
ENTRY:       ACCOUNT (CUSTOMER (PURCH-DATE) ),
             STOCK# (!PRODUCT),
             QUANTITY,
             PRICE,
             TAX,
             TOTAL,
             PURCH-DATE (DATE-MASTER),
             DELIV-DATE (DATE-MASTER);
CAPACITY:    12000;
```

where

set name is the data set name. It must be a valid IMAGE data name as defined in table 3-1.

DETAIL or D denotes a detail data set.

read class list is a group of user class numbers between 0 and 63, inclusive, separated by commas. User class numbers are described in Section II.

write class list is a group of user class numbers between 0 and 63, inclusive, separated by commas.

item name is the name of a data item defined in the item part. Each item defined as a search item must be a simple item. Up to 16 items may be search items. (See *master set name* for more information about search items.)

! denotes a primary path. Only one path in each detail data set can be designated as a primary path. If no path is designated as primary, the first unsorted path is the primary path by default. If all of the paths are sorted, the default primary path is the first sorted path.

master set name is the name of 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 set to the named master. Up to 16 search items can be defined for a detail data set. If no data item has a master name following it, the detail is not related to any master. In this case, the combined length of all data items in the data set must equal or exceed two words.

sort item name is the name of a detail data item of type U, K, or X which is part of the data set being defined. A sort item defines a sorted path. Each entry added to a chain of a sorted path will be linked logically in ascending order of the sort item values. If sort item is omitted, the path order is chronological, that is, new entries are linked to the end of chains. For performance reasons, sorted chains should be kept short. (Refer to page 2–7.)

maximum entry count is the maximum number of entries the data set can contain, the data set's capacity. It must be less than 2^{23} (8,388,608).

MASTER AND DETAIL SEARCH ITEMS

The master and detail search items that define a path between two data sets must have identical *type designators* and *sub-item lengths* when they are defined in the item part. Since the same data item name may appear in more than one data set, you may use the same data item name and definition for both the master and detail search items. For example, the data item ACCOUNT is used as the search item in both the CUSTOMER master and SALES detail data sets.

If you want to make a distinction between the search items, however, they may be defined separately. An example of this technique is found in the STORE data base. The search item DATE links the DATE-MASTER data set to the SALES data set through two paths, and two search items, PURCH-DATE and DELIV-DATE. These three data items look like this in the item part:

```
DATE,          X6;
DELIV-DATE,    X6  (/14);
PURCH-DATE,    X6  (11/14);
```

Each data item has type designator X and sub-item length 6. The item names, read class lists, and write class lists differ however.

Figure 3-5 at the end of this section contains the listing printed by the Schema Processor when the STORE data base schema is processed. Refer to this figure for examples of the schema parts.

DATA SET IDENTIFIERS

Like data items, data sets may be referenced by name or number. The data set number is determined by the set's position in the set part of the schema. It is more flexible to use data set names, however, in order to maintain program file independence.

SCHEMA PROCESSOR OPERATION

The Schema Processor is a program which accepts a *textfile* containing the schema as input, scans the schema and if no errors are detected, optionally produces a root file. The Schema Processor prints a heading, an optional list of the schema, and summary information on a *listfile.*

The Schema Processor executes in either MPE job or session mode. For further information about sessions and jobs, refer to the *MPE Commands Reference Manual.* In either case, you must use the MPE command:

```
:RUN DBSCHEMA.PUB.SYS
```

to initiate execution of the Schema Processor.

Table 3-5 lists the formal file designators and default actual file designators which the Schema Processor uses for *textfile* and *listfile.* The input/output devices to which $STDINX and $STDLIST refer depend upon the way your system is generated. However, $STDINX is the standard job or session input device and $STDLIST is the standard job or session output device.

Table 3-5. Schema Processor Files

FILE	USE	FORMAL FILE DESIGNATOR	DEFAULT ACTUAL FILE DESIGNATOR
textfile	Schema and Schema Processor commands	DBSTEXT	$STDINX
listfile	output listing	DBSLIST	$STDLIST

If you want to equate these files to some other actual file designator, you can use the MPE :FILE command. If a :FILE command is included in the job stream, you must inform the Schema Processor of this in the :RUN command in the following way:

```
:RUN DBSCHEMA.PUB.SYS;PARM=n
```

where

n = 1

if an actual file designator has been equated to DBSTEXT

n = 2

if an actual file designator has been equated to DBSLIST

n = 3

if actual file designators have been equated to both DBSTEXT and DBSLIST.

Table 3-6 shows sample combinations of RUN and FILE commands which can be used to initiate DBSCHEMA execution.

Table 3-6. RUN and FILE Commands, Examples

:RUN DBSCHEMA.PUB.SYS	Uses all default files. Prompts for lines of schema in session mode.
:FILE DBSTEXT=GEORGE :RUN DBSCHEMA.PUB.SYS;PARM=1	Processes schema from a user disc textfile named GEORGE.
:FILE DBSLIST;DEV=LP :RUN DBSCHEMA.PUB.SYS;PARM=2	Outputs the listing to a line printer.
:FILE DBSTEXT=GEORGE :FILE DBSLIST;DEV=LP :RUN DBSCHEMA.PUB.SYS;PARM=3	Processes schema from a user **textfile** named GEORGE; outputs the listing to a line printer.

Only the first 72 characters of each textfile record are processed.

If you request a root file, and the schema is error-free, it is created, given the same name as the one specified for the data base in the schema, initialized, and saved as a catalogued disc file.

CREATING THE TEXTFILE

A convenient method for creating the input file is to use the text editor, EDIT/3000, to enter the commands and schema in a disc file. Figure 3-2 illustrates this process in a sample session which also executes the Schema Processor. User input is underlined. (Refer to *EDIT/3000 Reference Manual* for information about the Editor.)

The steps followed in the sample in figure 3-2 are:

1. Initiate an MPE session by logging on with the appropriate user name and account.

2. Initiate text editor execution. Enter an Editor ADD command in response to the first prompt.

3. Enter Schema Processor commands and the schema itself into records of the Editor work file.

4. Save the work file in a disc file named SCHEMAB. Then terminate the Editor.

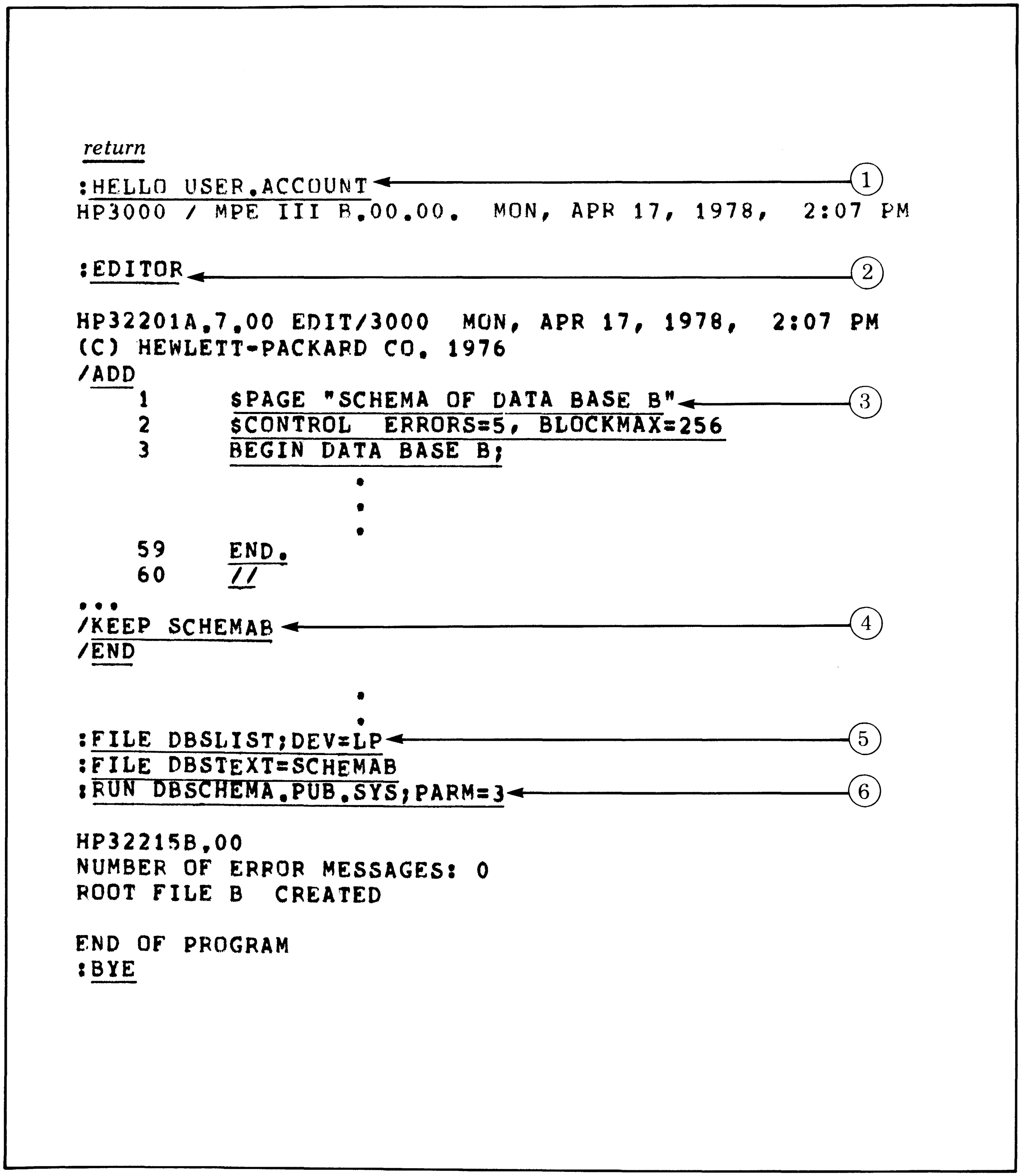
5. Use the :FILE command to equate the formal file designator DBSLIST to the line printer and DBSTEXT to the disc file SCHEMAB.

6. Initiate execution of DBSCHEMA and indicate that the *textfile* and *listfile* have been defined in :FILE commands. When the Schema Processor has finished processing the schema it prints the number of error messages and verifies that the root file has been created.

Figure 3-3 illustrates the order of commands and other input required when executing the Schema Processor in batch mode. The job can also be stored in a disc file and executed from a terminal.

THE DATA BASE CREATOR

The person who creates the root file is identified as the data base creator and can subsequently create and initialize the data base. To do so, the data base creator must log on with the same account, user name, and group that he or she used to create the root file and execute the IMAGE utility program DBUTIL. This program is described in Section VI.



```
return
:HELLO USER.ACCOUNT
HP3000 / MPE III B.00.00.   MON, APR 17, 1978,   2:07 PM

:EDITOR

HP32201A.7.00 EDIT/3000  MON, APR 17, 1978,  2:07 PM
(C) HEWLETT-PACKARD CO. 1976
/ADD
    1      $PAGE "SCHEMA OF DATA BASE B"
    2      $CONTROL  ERRORS=5, BLOCKMAX=256
    3      BEGIN DATA BASE B;
                         .
                         .
                         .
    59     END.
    60     //
...
/KEEP SCHEMAB
/END

                         .
                         .
:FILE DBSLIST;DEV=LP
:FILE DBSTEXT=SCHEMAB
:RUN DBSCHEMA.PUB.SYS;PARM=3

HP32215B.00
NUMBER OF ERROR MESSAGES: 0
ROOT FILE B  CREATED

END OF PROGRAM
:BYE
```

Figure 3-2. Sample Schema Creation Session

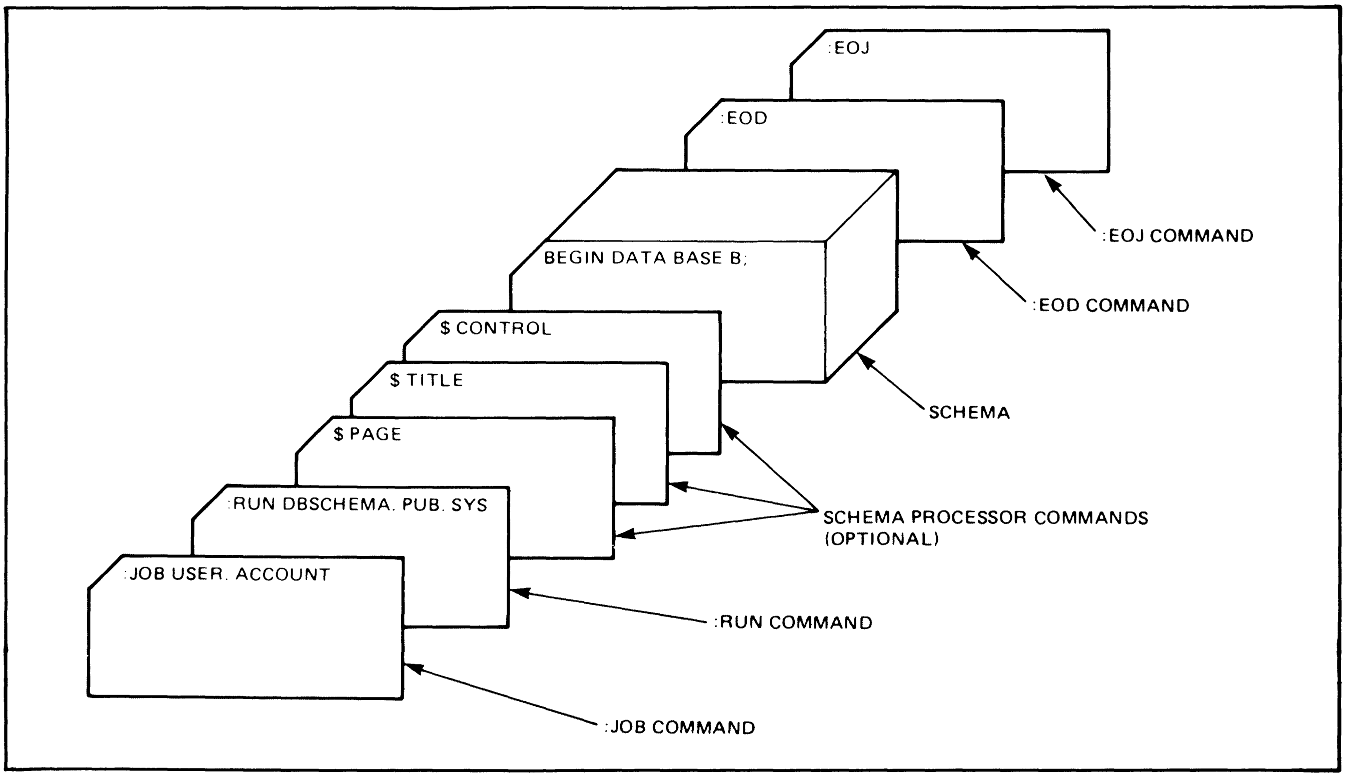


Figure 3-3. Schema Processor Batch Job Stream

SCHEMA PROCESSOR COMMANDS

IMAGE provides several commands which you may use anywhere in the schema to specify options available while processing the schema. The commands are: $PAGE, $TITLE, and $CONTROL. The $ must always be the first character of the record, immediately followed by the command name, which must be completely spelled out.

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

Command records may not contain comments.

CONTINUATION RECORDS

To continue a command to the next record, use an ampersand (&) as the last non-blank character in the current record. The following record must begin with a $. The records are combined and the $ and & are deleted and replaced by one blank character. A command name or parameter cannot be broken by &. Characters beyond the 72nd character of each record are ignored.

$PAGE

$PAGE COMMAND

The $PAGE command causes the *listfile* to eject to the top of the next page, print character-strings which you may optionally specify, and skip two more lines before continuing the listing.

The form of the $PAGE command is

$PAGE [["*character-string*"], . . .]

For example,

```
$PAGE     "STORE DATA BASE SCHEMA", " VERSION 3"
                 ↖                      ↗
                   character-string
```

where

character-string is a list of characters enclosed in quotes. When the command is executed, the quotes are stripped and the character-strings are concatenated. A quote mark within a character-string is specified by a pair of quotes.

The $PAGE command is effective only if the LIST option of the $CONTROL command is on. The LIST option is on by default until a $CONTROL command sets NOLIST. The $PAGE command itself is not listed.

The contents of the character-strings replace those specified by a previous $PAGE or $TITLE command. If no character-strings are specified, the character-strings specified in the preceding $PAGE or $TITLE command, if any, are printed at the top of the next page.

EXAMPLES

```
$PAGE "MASTER DATA SETS"&
$,"ACCOUNTING APPLICATION"

$PAGE
```

$TITLE

$TITLE COMMAND

The $TITLE command specifies a list of characters to be printed each time a heading is printed on a new page. It does not cause a page eject.

> The form of the $TITLE command is
>
> $TITLE [[“*character-string*”], . . .]

For example,

```
$TITLE “INVENTORY DATA BASE SCHEMA   B. J. BRINDISI”
                            ↑
                     character-string
```

where

character-string is a list of characters enclosed in quotes. When the command is executed, the quotes are stripped and the character-strings are concatenated. A quote mark within a character-string is specified by a pair of quotes.

The $TITLE command may be overridden by a subsequent $TITLE or $PAGE command. If no *character-string* is specified, no title is printed after the command is encountered until another $TITLE or $PAGE command specifies one.

EXAMPLE

```
$TITLE “ “ “QUICK” ” TEST DATA BASE”
```

$CONTROL

$CONTROL COMMAND

The $CONTROL command allows you to specify options in relation to processing the schema.

The form of the $CONTROL command is

```
$CONTROL  [LIST  ]   [,ERRORS=nnn]  [,LINES=nnnnn]  [,ROOT  ]
          [NOLIST]                                  [,NOROOT]

          [,BLOCKMAX=nnnn]  [,TABLE  ]
                            [,NOTABLE]
```

For example,

```
$CONTROL NOLIST, ERRORS= 5, LINES=62, NOROOT, BLOCKMAX=256,TABLE
                         ↗         ↗                     ↑
                      nnn      nnnnn                   nnnn
```

where

LIST — causes each source record of the schema to be printed on the *listfile.*

NOLIST — specifies that only source records with errors be printed on the listfile. An error message is printed after these records.

ERRORS=*nnn* — sets the maximum number of errors to *nnn*. If more than *nnn* errors are detected, the Schema Processor terminates. *nnn* may have a value between 0 and 999, inclusive. The default value is 100.

LINES=*nnnnn* — sets the number of lines per page on the listfile to *nnnnn* which can be between 4 and 32767, inclusive. The default value is 60 if listfile is a line printer and 32767 if it is not one.

ROOT — causes the Schema Processor to create a root file if no errors are detected in the schema.

NOROOT — prevents the Schema Processor from creating a root file.

BLOCKMAX=*nnnn* — sets the maximum physical block length (in words) for any data set in the data base. *nnnn* may have a value between 128 and 2048, inclusive. The default value is 512. This is an important parameter and is discussed in greater detail below.

TABLE — causes the Schema Processor to write a table of summary information about the data sets to the *listfile* device if no errors are detected.

NOTABLE — suppresses the TABLE option.

The default parameters are underlined. If no $CONTROL command is used the results are the same as if the following $COMMAND command is used:

```
$CONTROL LIST,ERRORS=100,LINES=60, ROOT,BLOCKMAX=512,TABLE
                                ↑
                           (or 32767)
```

The parameters may be placed in any order but must be separated by commas.

SELECTING THE BLOCK SIZE

The data set records are transferred from the disc to memory in blocks. (The block format is described in Section VII.) When you specify a maximum block size with the $CONTROL command you should consider:

- efficient disc space utilization
- minimum disc access
- program execution time which can be affected by the size of a privileged data segment in which IMAGE maintains a Data Base Control Block. (Refer to Section IV for a definition of the DBCB.) Buffers in the DBCB must be as large as the largest block of the data base, therefore, the larger the block, the larger this data segment must be.

The Schema Processor determines the number of data records which fit in a block. Larger blocks minimize disc access by enabling the transfer of more records at one time. In selecting a block size, the following considerations may apply:

- If the applications using the data base will be run as batch jobs at times when few other users are competing for system resources, particularly memory space, you may choose to use large blocks. This will reduce the frequency of disc access if an application is accessing data sets serially, or along chains whose members are physically contiguous or close.
- If the application programs are large and will be run while many users are operating in session mode, large blocks and the resulting large DBCB data segment may cause the program to execute more slowly since a larger area of memory is required to execute the program. In this case, you may need to decrease the block size. If the application programs are small, this may not be necessary.

Other factors may depend on the application requirements and a certain amount of tuning is sometimes necessary to determine the best block size. In general, the default block size of 512 words yields reasonable performance and should be changed only with good reason.

SCHEMA PROCESSOR OUTPUT

The Schema Processor prints the following heading on the first page of the listing:

```
                      product identification          product name
                                |                          /
PAGE 1       HEWLETT-PACKARD 32215B.00       IMAGE/3000   MON, APR 3, 1978,   4:32 PM
```

If your standard output device ($STDLIST) is different from *listfile*, an abbreviated product identification is also printed on $STDLIST. Subsequent pages of *listfile* are headed by a page number, the data base name if it has been encountered, and the title most recently specified by a $TITLE or $PAGE command.

If the LIST option is active, a copy of each record of the schema is sent to the *listfile*. However, if the *textfile* and *listfile* are the same, as for example they are when you enter the schema source from your terminal in session mode, the records are not listed. If you are entering the schema in this way, the Schema Processor prompts for each line of input with a >.

SUMMARY INFORMATION

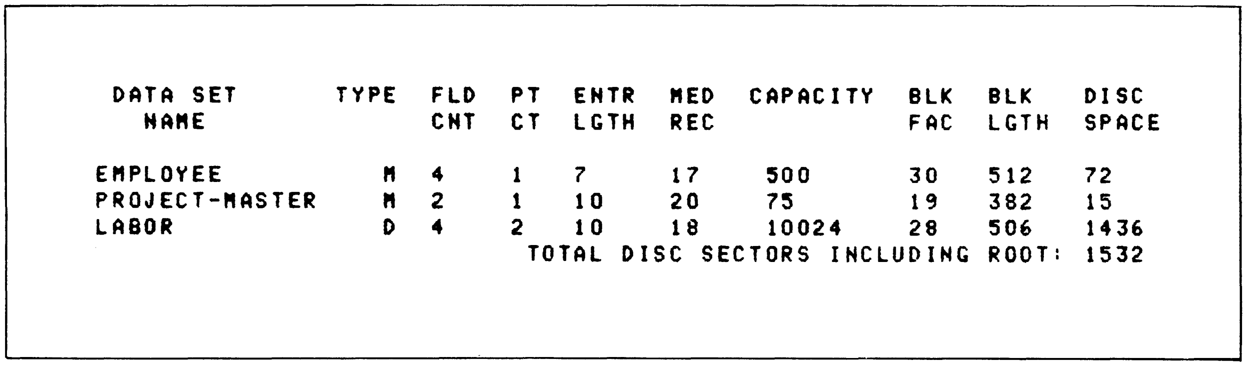
After the entire schema has been scanned, several types of summary information may be printed on the listfile.

- If not all of the items defined in the item part are referenced in the set part, and if no errors are encountered, the message:

  UNREFERENCED ITEMS: *list of items*

  is printed to the *listfile*. The list includes all items defined but not referenced in a data set. Although they are not considered errors, these extraneous items should be removed to reduce the size of the tables in the root file and the size of the extra data segment used by the library procedures.

- If no errors are detected in the schema and if the TABLE option has been selected, the Schema Processor prints a table of summary information about the data sets. Figure 3-4 contains a sample printout of this information. Table 3-7 describes the information contained in the summary. The NOTABLE parameter of the $CONTROL command suppresses printing of this table.



```
   DATA SET        TYPE  FLD  PT  ENTR  MED  CAPACITY  BLK  BLK   DISC
     NAME                CNT  CT  LGTH  REC            FAC  LGTH  SPACE

  EMPLOYEE           M   4    1   7     17    500      30   512   72
  PROJECT-MASTER     M   2    1   10    20    75       19   382   15
  LABOR              D   4    2   10    18    10024    28   506   1436
                                TOTAL DISC SECTORS INCLUDING ROOT:  1532
```

Figure 3-4. Data Set Summary Table

Table 3-7. Data Set Summary Table Information

DATA SET NAME	The name of the data set.
TYPE	A for automatic, M for manual, or D for detail
FLD CNT	The number of data items in each entry of the data set.
PT CT	Path count. For a master data set, this is the number of paths specified for the data set search item. For a detail data set, it is the number of search items defined for each entry of the data set.
ENTR LGTH	The length in words of the data portion of the data entry (not including any of the IMAGE/3000 pointers or other structure information associated with a data entry).
MED REC	The total length in words of a media record of the data set. This length includes the entry length plus any of the IMAGE/3000 pointers associated with the data entry. Media records are discussed in Section VI.
CAPACITY	The maximum number of entries allowed in the data set. For detail data sets, this number may differ from the number of entries specified in the schema itself, because the capacity of each detail is adjusted to represent an even multiple of the blocking factor (see below).
BLK FAC	The number of media records which are blocked together for transfer to and from the disc.
BLK LGTH	The total length in words of the physical block as defined in BLK FAC. This includes the media records and a bit map. Bit maps are discussed in Section VI.
DISC SPACE	The amount of disc space (in 128-word sectors) occupied by the MPE file containing the data set.
TOTAL DISC SECTORS INCLUDING ROOT: **nnnn**	The total number of 128-word disc sectors which will be occupied by the data base, when created using the DBUTIL program.

- Two lines of summary totals are printed on the *listfile*. For example:

```
NUMBER OF ERROR MESSAGES: 0
ITEM NAME COUNT: 22        DATA SET COUNT: 6
```

The error count includes both errors in the schema and in the Schema Processor commands. The error count is also sent to $STDLIST, if it is different from the *listfile*.

- If no schema syntax or logical errors are encountered, a third line is printed. The form of this line is:

ROOT LENGTH: *r*   BUFFER LENGTH: *b*   TRAILER LENGTH: *t*

ROOT LENGTH is the length in words of the body of the root file. BUFFER LENGTH is the length in words of each of the data buffers which IMAGE allocates in an extra data segment (the DBCB) for use in transferring data set blocks to and from disc. TRAILER LENGTH is the length in words of an area in the extra data segment used by IMAGE to transfer information to and from a calling program's stack.

- If no errors are detected and the ROOT option is active, the following message is sent to the *listfile:*

  ROOT FILE *data base name* CREATED

  data base name is the name given in the BEGIN DATA BASE statement in the schema.

SCHEMA ERRORS

When the Schema Processor detects an error it prints a message to the *listfile.* If the LIST option is active, it is printed immediately after the offending statement. If NOLIST is active, the current line of the schema is printed and then the error message.

Schema Processor error messages are explained in Appendix A. The root file is not created if any of the listed errors are detected. However, the Schema Processor attempts to continue checking the schema for logical and syntactical correctness.

One error may obscure detection of subsequent errors, particularly if it occurs early in a data set. It may be necessary to process the schema again after the error is corrected to find subsequent errors. Conversely, some errors early in the schema can generate subsequent apparent errors which will disappear after the original error has been corrected.

If schema errors prohibit creation of the root file, the following message is sent to the *listfile,* and to $STDLIST if it is not the same as the *listfile:*

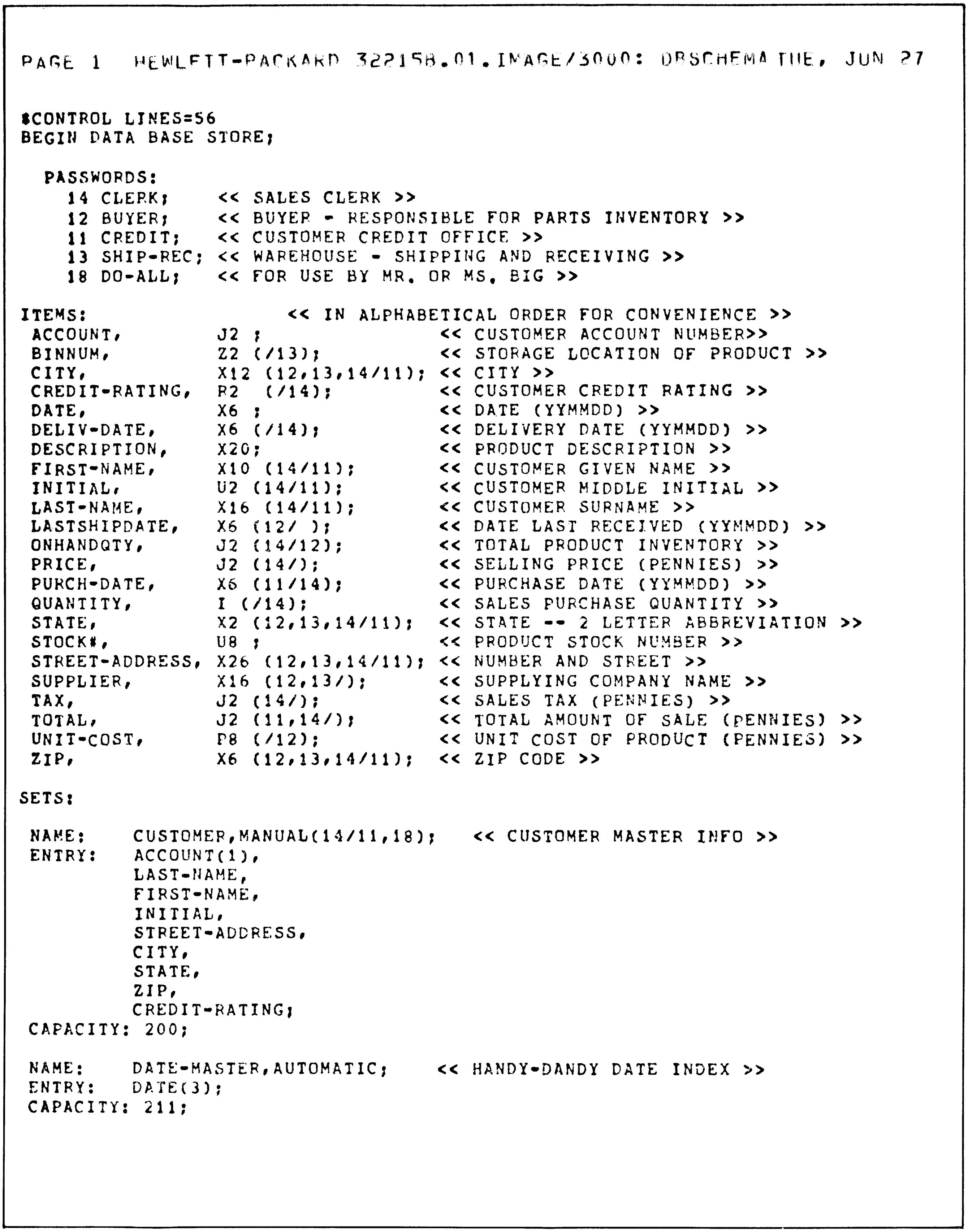
PRECEDING ERRORS — NO ROOT FILE CREATED.

A few conditions, including the number of errors exceeding the total number allowed, cause immediate termination of the Schema Processor without the normal summary lines. In this case, the following message is printed:

SCHEMA PROCESSING TERMINATED.

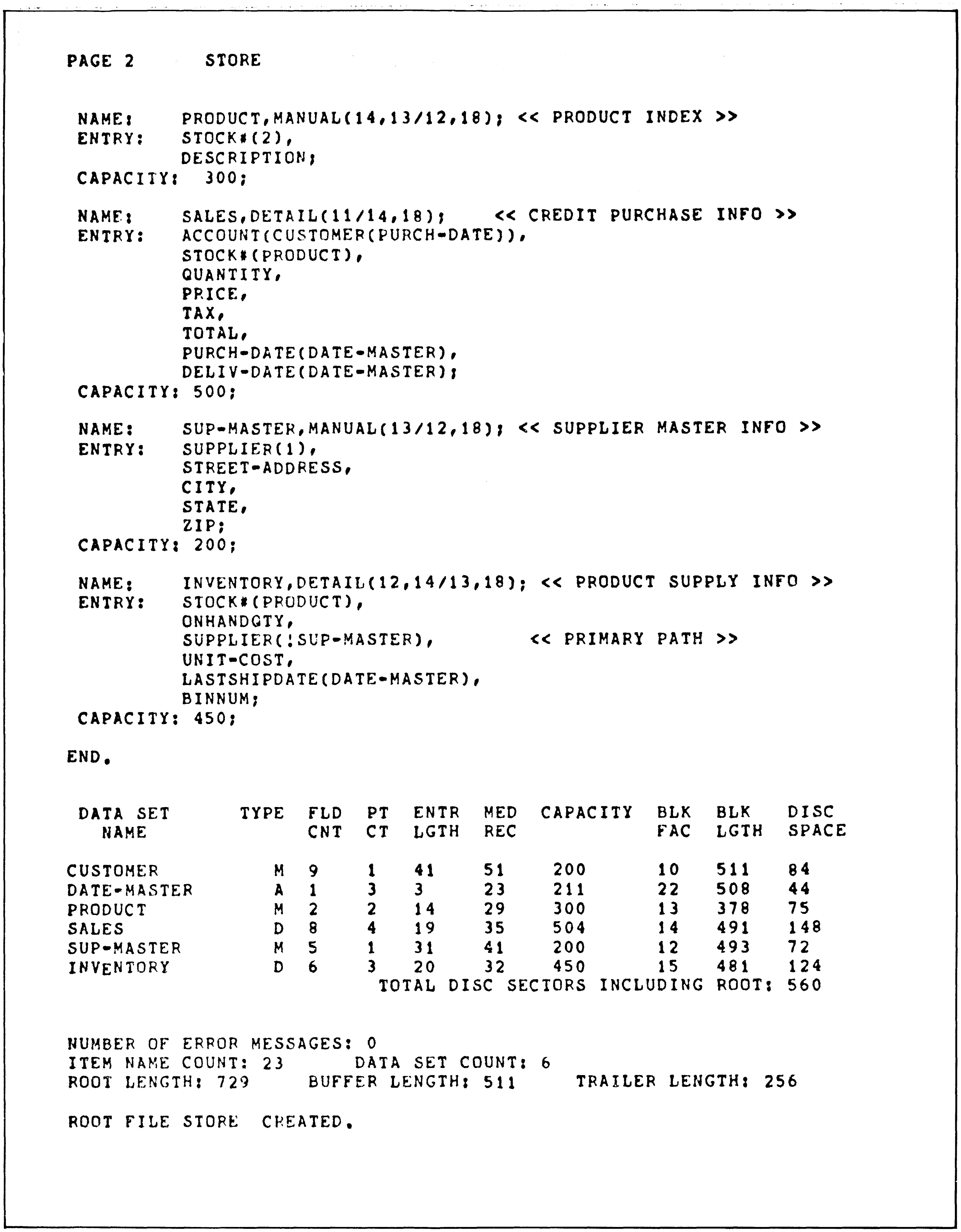
SCHEMA PROCESSOR EXAMPLE

Figure 3-5 contains the *listfile* output printed when the schema of the sample STORE data base is processed. The data base has 5 passwords and contains 23 data item definitions and 6 data set definitions. The Schema Processor summary information is printed following the schema.



```
PAGE 1   HEWLETT-PACKARD 32215B.01.IMAGE/3000: DBSCHEMA TUE, JUN 27


$CONTROL LINES=56
BEGIN DATA BASE STORE;

  PASSWORDS:
    14 CLERK;     << SALES CLERK >>
    12 BUYER;     << BUYER - RESPONSIBLE FOR PARTS INVENTORY >>
    11 CREDIT;    << CUSTOMER CREDIT OFFICE >>
    13 SHIP-REC;  << WAREHOUSE - SHIPPING AND RECEIVING >>
    18 DO-ALL;    << FOR USE BY MR. OR MS. BIG >>

ITEMS:                  << IN ALPHABETICAL ORDER FOR CONVENIENCE >>
 ACCOUNT,         J2 ;                 << CUSTOMER ACCOUNT NUMBER>>
 BINNUM,          Z2 (/13);            << STORAGE LOCATION OF PRODUCT >>
 CITY,            X12 (12,13,14/11);   << CITY >>
 CREDIT-RATING,   R2  (/14);           << CUSTOMER CREDIT RATING >>
 DATE,            X6 ;                 << DATE (YYMMDD) >>
 DELIV-DATE,      X6 (/14);            << DELIVERY DATE (YYMMDD) >>
 DESCRIPTION,     X20;                 << PRODUCT DESCRIPTION >>
 FIRST-NAME,      X10 (14/11);         << CUSTOMER GIVEN NAME >>
 INITIAL,         U2 (14/11);          << CUSTOMER MIDDLE INITIAL >>
 LAST-NAME,       X16 (14/11);         << CUSTOMER SURNAME >>
 LASTSHIPDATE,    X6 (12/ );           << DATE LAST RECEIVED (YYMMDD) >>
 ONHANDQTY,       J2 (14/12);          << TOTAL PRODUCT INVENTORY >>
 PRICE,           J2 (14/);            << SELLING PRICE (PENNIES) >>
 PURCH-DATE,      X6 (11/14);          << PURCHASE DATE (YYMMDD) >>
 QUANTITY,        I (/14);             << SALES PURCHASE QUANTITY >>
 STATE,           X2 (12,13,14/11);    << STATE -- 2 LETTER ABBREVIATION >>
 STOCK#,          U8 ;                 << PRODUCT STOCK NUMBER >>
 STREET-ADDRESS,  X26 (12,13,14/11);   << NUMBER AND STREET >>
 SUPPLIER,        X16 (12,13/);        << SUPPLYING COMPANY NAME >>
 TAX,             J2 (14/);            << SALES TAX (PENNIES) >>
 TOTAL,           J2 (11,14/);         << TOTAL AMOUNT OF SALE (PENNIES) >>
 UNIT-COST,       P8 (/12);            << UNIT COST OF PRODUCT (PENNIES) >>
 ZIP,             X6 (12,13,14/11);    << ZIP CODE >>

SETS:

 NAME:    CUSTOMER,MANUAL(14/11,18);    << CUSTOMER MASTER INFO >>
 ENTRY:   ACCOUNT(1),
          LAST-NAME,
          FIRST-NAME,
          INITIAL,
          STREET-ADDRESS,
          CITY,
          STATE,
          ZIP,
          CREDIT-RATING;
 CAPACITY: 200;

 NAME:    DATE-MASTER,AUTOMATIC;     << HANDY-DANDY DATE INDEX >>
 ENTRY:   DATE(3);
 CAPACITY: 211;
```

Figure 3-5. STORE Data Base Schema



```
PAGE 2      STORE


 NAME:     PRODUCT,MANUAL(14,13/12,18); << PRODUCT INDEX >>
 ENTRY:    STOCK#(2),
           DESCRIPTION;
 CAPACITY:  300;

 NAME:     SALES,DETAIL(11/14,18);    << CREDIT PURCHASE INFO >>
 ENTRY:    ACCOUNT(CUSTOMER(PURCH-DATE)),
           STOCK#(PRODUCT),
           QUANTITY,
           PRICE,
           TAX,
           TOTAL,
           PURCH-DATE(DATE-MASTER),
           DELIV-DATE(DATE-MASTER);
 CAPACITY: 500;

 NAME:     SUP-MASTER,MANUAL(13/12,18); << SUPPLIER MASTER INFO >>
 ENTRY:    SUPPLIER(1),
           STREET-ADDRESS,
           CITY,
           STATE,
           ZIP;
 CAPACITY: 200;

 NAME:     INVENTORY,DETAIL(12,14/13,18); << PRODUCT SUPPLY INFO >>
 ENTRY:    STOCK#(PRODUCT),
           ONHANDQTY,
           SUPPLIER(!SUP-MASTER),       << PRIMARY PATH >>
           UNIT-COST,
           LASTSHIPDATE(DATE-MASTER),
           BINNUM;
 CAPACITY: 450;

END.


 DATA SET       TYPE  FLD  PT  ENTR  MED  CAPACITY  BLK  BLK   DISC
   NAME               CNT  CT  LGTH  REC            FAC  LGTH  SPACE

CUSTOMER          M   9    1   41    51    200      10   511   84
DATE-MASTER       A   1    3   3     23    211      22   508   44
PRODUCT           M   2    2   14    29    300      13   378   75
SALES             D   8    4   19    35    504      14   491   148
SUP-MASTER        M   5    1   31    41    200      12   493   72
INVENTORY         D   6    3   20    32    450      15   481   124
                          TOTAL DISC SECTORS INCLUDING ROOT:   560


NUMBER OF ERROR MESSAGES: 0
ITEM NAME COUNT: 23      DATA SET COUNT: 6
ROOT LENGTH: 729     BUFFER LENGTH: 511     TRAILER LENGTH: 256

ROOT FILE STORE  CREATED.
```

Figure 3-5. STORE Data Base Schema (Continued)

USING THE DATA BASE

SECTION IV

After the data base is designed and the root file has been created, application programs can be written that will be used to enter and use the data. Programs written in COBOL, FORTRAN, SPL, or BASIC gain access to the data base through calls to IMAGE procedures. RPG programs contain specifications used by the Report Program Generator to make calls to the IMAGE procedures for you. Table 4-1 contains a list of the procedures with a general description of their function. Specific information about procedure calls, parameters, and status information is given later in this section.

Table 4-1. IMAGE Procedures

PROCEDURE	FUNCTION
DBOPEN	Initiates access to a data base. Sets up user's access mode and user class number for the duration of the process.
DBPUT	Adds new entries to a data set.
DBFIND	Locates the first and last entries of a data chain in preparation for access to entries in the chain.
DBGET	Reads the data items of a specified entry.
DBUPDATE	Updates or modifies the values of data items that are not search or sort items.
DBDELETE	Deletes existing entries from a data set.
DBLOCK	Locks one or more data entries, a data set, or an entire data base (or any combination of these) temporarily to allow the process calling the procedure to have exclusive access to the locked entities.
DBUNLOCK	Releases those locks obtained with previous calls to DBLOCK.
DBCLOSE	Terminates access to a data base or a data set, or resets the pointers of a data set to their original state.
DBINFO	Provides information about the data base being accessed, such as the name and description of a data item.
DBEXPLAIN	Examines status information returned by an IMAGE procedure that has been called and prints a multi-line message on the $STDLIST device.
DBERROR	Supplies an English language message that interprets the status information set by any callable IMAGE procedure. The message is returned to the calling program in a buffer.

NOTE

Before the application programs can be executed, the data base must be created using the DBUTIL IMAGE utility program described in Section VI.

OPENING THE DATA BASE

Before you can gain access to the data, the process you are running must open the data base with a call to the DBOPEN procedure. In opening a data base, DBOPEN establishes an access path between the data base and your program by:

- verifying your right to use the data base under the security provisions provided by the MPE file system and the IMAGE user class/password scheme
- determining that the access mode you have requested in opening the data base is compatible with the access modes of other users currently using the data base
- opening the root file and constructing the control blocks to be used by all other IMAGE procedures when they are executed. The root file remains open until the data base is closed.

DATA BASE CONTROL BLOCK

A Data Base Control Block (DBCB) is a table of data base relative information residing in a privileged extra data segment. There is exactly *one* DBCB for each open data base regardless of the number of concurrent access paths to the data base. The DBCB contains all the static and dynamic global information; that is, it contains all information which is not unique to a particular access path.

IMAGE creates the DBCB for a particular data base when the first user's process calls the DBOPEN procedure to open the data base and establish an access path. The global information for access paths to the same data base (established by subsequent calls to DBOPEN) will be maintained in the same DBCB. The DBCB is destroyed (released) when the last concurrent user closes the data base.

The information in the DBCB is initially derived from the root file. In addition to tables describing the various components of the data base, the DBCB contains all buffers and work areas used by the IMAGE procedures. IMAGE may modify the size and contents of various areas within the DBCB to reflect the addition of access paths as more users open the data base. All IMAGE procedures operating on a particular data base reference the same DBCB. (DBERROR and DBEXPLAIN are exceptions; they do not reference the DBCB.)

USER LOCAL CONTROL BLOCK

A User Local Control Block (ULCB) is a table of information which is specific to a particular user's access to a particular data base. There is one ULCB for each access path established for each IMAGE data base. In other words, a unique ULCB is created each time DBOPEN is successfully called. A ULCB contains most of the static and dynamic data which is unique to a specific access path. (Some of this data is in the DBCB.) For example, a user's current record numbers, current lists, and security information are kept in his or her ULCB. The privileged extra data segment containing the ULCB is associated with the user's process.

The initial contents of the ULCB come from the DBCB and the root file. IMAGE procedures called after DBOPEN combine information in the user's ULCB with information in the global DBCB in order to process the user's request.

The ULCB is released when the user's process calls DBCLOSE to close the data base.

PASSWORDS

When you open the data base you must provide a valid password to establish your *user class number.* If you do not provide one, you will be granted user class number 0. If you are the data base creator and supply a semicolon as a password, the number 64 is used to grant you unlimited data base access privileges. Passwords and user classes are discussed in Section II.

ACCESS MODES

There are eight different access modes available, each mode determines the type of operation that you can perform on the data base as well as the types of operations other users can perform simultaneously. To simplify the definition of the various access modes, the following terminology is used:

- read access allows the user to locate and read data entries.
- update access allows read access and, in addition, allows the user to replace values in all data items except search and sort items.
- modify access allows update access and, in addition, allows the user to add and delete entries.

The procedures that can be used with each type of access are:

- **read** — DBFIND and DBGET
- **update** – DBFIND, DBGET, and DBUPDATE
- **modify** – DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE

Table 4-2 summarizes the type of access granted in each access mode, provided the MPE security provisions and your password permit it. Access modes 3 and 7 provide exclusive access to the data base; all other modes allow shared access.

Table 4-2. Access Mode Summary

ACCESS MODE	TYPE OF ACCESS GRANTED	CONCURRENT ACCESS ALLOWED	SPECIAL REQUIREMENTS
1	modify	modify (with locking)	Locking must be used for update or modify.
2	update	update	
3	modify	none	
4	modify	read	
5	read	modify (with locking)	IMAGE does not require locking but it should be used to coordinate access with users who are modifying.
6	read	modify	
7	read	none	
8	read	read	

CONCURRENT ACCESS MODES. A data base can only be shared in certain well-defined environments. The access mode specified when a process opens a data base must be acceptable for the environment established by others who are already using the data base. Here is a summary of the acceptable environments:

- multiple mode 1 and mode 5 users
- multiple mode 6 and mode 2 users
- multiple mode 6 users and one mode 4 user
- multiple mode 6 and mode 8 users
- one mode 3 user
- one mode 7 user.

Subsets of these environments are also allowed. For example, there may be all mode 6 users or all mode 8 users. There may be one mode 1 user or all mode 5 users and so forth.

If a mode 3 or mode 7 user is currently accessing the data base, it cannot be opened until that user closes the data base. This is true any time an attempt is made to open a data base in a mode which is not compatible with the modes of others using the data base.

DATA BASE OPERATIONS. The descriptions below explain in detail exactly what occurs when a data base is opened in a particular mode. Locking is available in all modes. In the discussion that follows, brief suggestions are given as to when locking may be used. Refer to the discussion of the locking facility for more information.

- ACCESS MODE 1. The data base is opened for shared modify access. Opening in mode 1 succeeds only if all other current users of the data base have access modes 1 or 5.

  All IMAGE procedures are available in this mode. However, a program must obtain temporary exclusive control of the data entries before calling any procedure that changes them, such as DBUPDATE, DBPUT, or DBDELETE. In this way, changes to the data base are synchronized and carried out properly. This exclusive control must subsequently be relinquished to permit other access mode 1 or mode 5 users to access these entries. Acquiring and relinquishing is referred to as locking and unlocking, respectively. These functions are supplied by the IMAGE library procedures, DBLOCK and DBUNLOCK. The locking requirements may be met by locking the affected entries, the sets containing the entries, or the whole data base.

  A mode 1 (and mode 5) user who has all or part of the data base locked is assured that no concurrent user is modifying that part of the data base.

  It is possible to read entries in the data base using calls to DBFIND and DBGET without locking but the calling program must provide for the possibility that another process may be simultaneously modifying the data base. This can result in an entry being deleted from a chain which the calling program is reading.

- ACCESS MODE 2. The data base is opened for shared update access. The opening succeeds only if all current users of the data base have access modes 2 and 6. All IMAGE procedures are available to the mode 2 user except DBPUT and DBDELETE which are permanently disabled in this mode. Therefore, the mode 2 user is able to read and update data entries but is not permitted to add or delete data entries in any data set.

  The programmer must be aware of the possibility that other mode 2 users are simultaneously updating data entries. In many applications, it may be possible to arrange for each user process to update unique data entries or data items so that the data base will correctly reflect all changes, even data items in the same entry updated by different processes. On the other hand, if two or more processes update the same data items of the same entry, the data base will reflect only the latest values. Locking may be used, if desired, to coordinate update sequences to an entry or to coordinate with mode 6 readers.

- ACCESS MODE 3. The data base is opened for exclusive modify access. If any other users are accessing the data base it cannot be opened in this mode. All IMAGE procedures are available to the mode 3 user. No other concurrent process is permitted to gain any type of access to the data base.

- ACCESS MODE 4. The data base is opened for semi-exclusive modify access. Only one mode 4 user can access the data base and all other current users must be in mode 6 (read only). The mode 4 user is permitted to call any IMAGE procedure and has complete control over data base content. This mode differs from mode 3 only in that other read-only users are permitted concurrent access to the data base. Locking may be used to coordinate with mode 6 readers.

- ACCESS MODE 5. The data base is opened for shared read access. All other concurrent users must be in mode 1 or mode 5. Mode 5 operates in exactly the same way as mode 1 except the procedures that alter the data base, DBUPDATE, DBPUT, and DBDELETE, are disabled for the mode 5 user. Locking can be used, if desired, to ensure that data is not being modified while you are reading it.

- ACCESS MODE 6. The data base is opened for shared read access. Concurrent users must be in mode 2, 4, 6, or 8. This mode can also be used while the data base is being stored with the IMAGE utility program, DBSTORE. Some of these modes are incompatible with each other as shown in the discussion of concurrent access modes above. All IMAGE procedures that alter the data base are disabled. Locking can be used to synchronize with users who are concurrently updating.

  Mode 5 and 6 are appropriate for inquiry-type applications if they can tolerate the possibility of data base modifications taking place simultaneously, since mode 1, 2, and 4 users can make such changes.

- ACCESS MODE 7. The data base is opened for exclusive read access. No other users may access the data base concurrently. Mode 7 operates in exactly the same way as mode 3 except the procedures that alter the data base are disabled for the mode 7 user.

- ACCESS MODE 8. The data base is opened for shared read access. Concurrent users must be in mode 6 or 8 or using the IMAGE utility DBSTORE. IMAGE procedures that alter the data base are not permitted. Since mode 8 allows only concurrent readers, a user program with this access mode can be assured that the data base values it reads are unchanging.

SELECTING AN ACCESS MODE. When deciding which access mode to use, two important considerations are:

- Use the least capability that will accomplish the task. For example, select a read only access mode (5, 6, 7, or 8) if the program does not alter the data base in any way. In these modes, the data base files are opened by IMAGE for input only. This type of access is more generally available than input-output access because MPE security provisions on concurrent data base use may prevent input-output access but allow input only access. Furthermore, files opened only for input access are not included on incremental MPE SYSDUMP tapes and therefore system back-up time may be reduced by opening the data base for read only access. (Refer to the *System Manager/System Supervisor Manual* for more information about SYSDUMP.)

- Allow concurrent users as much capability as is consistent with successful completion of the task. If the task is merely browsing through the data base, producing a quick report, or accessing an unchanging portion of the data base, choose a mode which allows concurrent users to make data base modifications to other parts of the data base. Allowing concurrent read-only access (modes 2, 4, and 8) may be appropriate in many situations. For programs that must be assured there will be no concurrent structural changes but can tolerate simultaneous updates to entries, mode 2 may be particularly suitable. Locking may be used to control simultaneous updates to a data entry. If it is absolutely necessary to make structural changes to a data base from concurrent multiple processes, modes 1 and 5 must be used. Fully exclusive operation (modes 3 and 7) are available if needed.

The following mode selection guidelines are organized according to the task to be performed. For some tasks, one of several modes may be selected depending on the concurrent activity allowed with each mode.

- Programs that perform all data base operations, including adding and deleting entries, should open with mode 1, 3, or 4. Choose mode:

  1 if it is necessary to allow other processes to add and delete entries simultaneously. In this case, the affected parts of the data base must be locked while performing updates, additions, or deletions.

  4 if exclusive ability to change the data base is required but it is possible to allow mode 6 processes to read the data base while changes are being made.

  3 if the program must have exclusive access.

- Programs that locate, read and replace data in existing entries but do not need to add or delete any entries, and do not want any other processes to do so, should open the data base in mode 2. Locking can be used to coordinate updates.

- Programs that only locate and read or report on information in the data base should open with one of the read only modes. In this case, the mode selected depends upon either the type of process running concurrently or the need for an unchanging data base while the program is running. Choose mode:

  5 if concurrent processes will operate in modes 1 or 5. Parts or all of the data base may optionally be locked to prevent concurrent changes during one or more read operations.

  6 if it is not important what other processes are doing to the data base. In this case, mode 2 processes can replace entries, one mode 4 user can replace, add or delete entries, or mode 6 or mode 8 users can read entries while the program is using the data base.

  8 if the data base must not change while the program is accessing it.

  7 if the program must have exclusive access to the data base.

DYNAMIC LOCKING. Refer to the discussion of locking and unlocking later in this section for some special considerations.

ENTERING DATA IN THE DATA BASE

Data is added to the data base, one entry at a time, using the DBPUT procedure. You may add data entries to manual master and detail data sets. Entries are automatically added to automatic master data sets when you add entries to the associated detail data sets.

To add an entry, you specify the data set name, a list of data items in the set, and the name of a buffer containing values for these items. Values must be supplied for search and sort items but are optional for other data items in the entry. If no value is supplied, the data item value is set to binary zeroes.

SEQUENCE FOR ADDING ENTRIES

Before you can add an entry to a detail data set indexed by a manual master data set, the manual master must contain an entry with a search item value equal to the one you intend to put in the detail. If more than one manual master is used to index the detail, entries which have a search item value identical to the detail search item value for the same path must exist in each master. To illustrate, consider the STORE data base again. Figure 4-1 contains sample data entries in four of the STORE data sets.

Before the SALES data entry can be added to the data set, the CUSTOMER manual master data set must contain an entry with ACCOUNT equal to 12345678 since ACCOUNT is the search item used to index the SALES detail. Similarly, the SALES data set is indexed by the PRODUCT manual master through the STOCK# search item, so the entry with STOCK# equal to 35624AB3 must be added to PRODUCT before a sales transaction for that STOCK# can be entered in SALES.

Once the entry for customer account 12345678 has been entered, the next sales transaction can be entered in the SALES detail set without changing the CUSTOMER master. This entry will be chained to the previous entry for the account. If a different customer buys a bicycle tire pump, the PRODUCT data set will not require any additional entries, but if the customer's account is not yet in the CUSTOMER data set it must be added before entering the sales transaction in SALES.

When the entry for account 12345678 and stock number 35624AB3 is added to SALES, IMAGE automatically adds entries to the DATE-MASTER with a DATE item value of 92775 and 92875 if such entries do not already exist. If the entries do exist, each chain head is modified to include the entry added to the chain.

ACCESS MODE AND USER CLASS NUMBER

An entry cannot be added to a data set unless the user class number established when the data base is opened grants this capability. The user class number must be in the data set write class list.

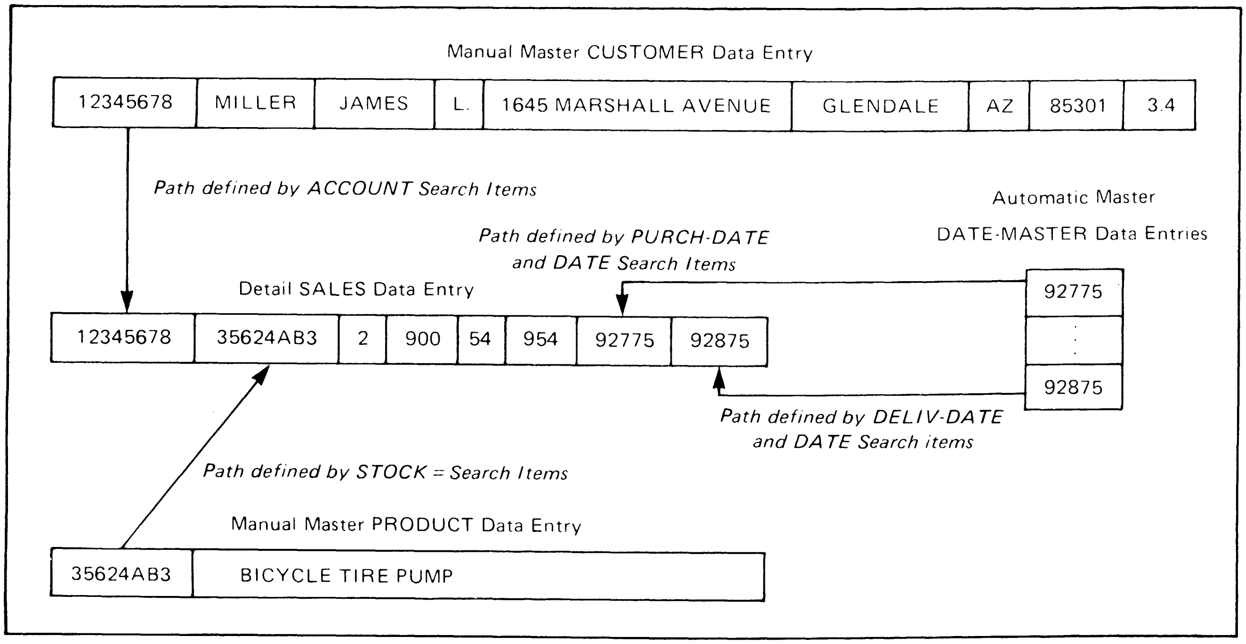


Figure 4-1. Sample Data Entries from STORE Data Base

The data base must also be opened with an access mode allowing entries to be added. These access modes are 1, 3, and 4. If it is opened with access mode 1, the DBLOCK procedure must be used to establish a lock covering the entry to be inserted. For detail data sets, this may be a data entry, data set, or data base lock. For manual master data sets, this must be a data set or data base lock.

Note that the locking mechanism will accept a request to lock a data entry that does not yet exist, therefore, you may lock a data entry before you add it.

SEARCH ITEMS

IMAGE performs checks on the values of search items before adding an entry to a data set. If the data set is a manual master, IMAGE verifies that the search item value is unique for the set, that no entry currently contains a search item with the same value. If the data set is a detail, IMAGE verifies that the value of each search item forming a path with a manual master has a matching value in that master. It also checks that there is room to add an entry to any automatic master data sets linked to the detail if a matching search item value does not exist.

READING THE DATA

When you read data from the data base you specify which data set and which entry in that data set contains the information you want. If the user class number with which you opened the data base grants you read access, you may read the entire entry or specific data items from the entry. You specify the items to be read and the array where the values should be stored. You can read items or entries in any access mode if your user class grants read access to the data element.

To understand the various ways in which you can select the data entry to be read, it is important to know a little about the data set structure. Each data set consists of one disc file and each data entry is a logical record in that file. Each entry is identified by the relative record number in which it is stored. The first record in the data set is record number 1 and the last is record number *n* where *n* is the capacity of the data set.

At any given time, a record may or may not contain an entry. IMAGE maintains internal information indicating which records of a data set contain entries and which do not.

CURRENT PATH

IMAGE maintains a *current path* for each detail data set and each accessor (access path). The current path is established by the DBFIND procedure, or if no call has been made to this procedure, it is the primary path for the data set. Each time an entry is read, no matter what read method is used, IMAGE saves the entry's backward and forward chain pointers for the current path. For more information about how the current path is used, refer to the discussion of chained access later in this section.

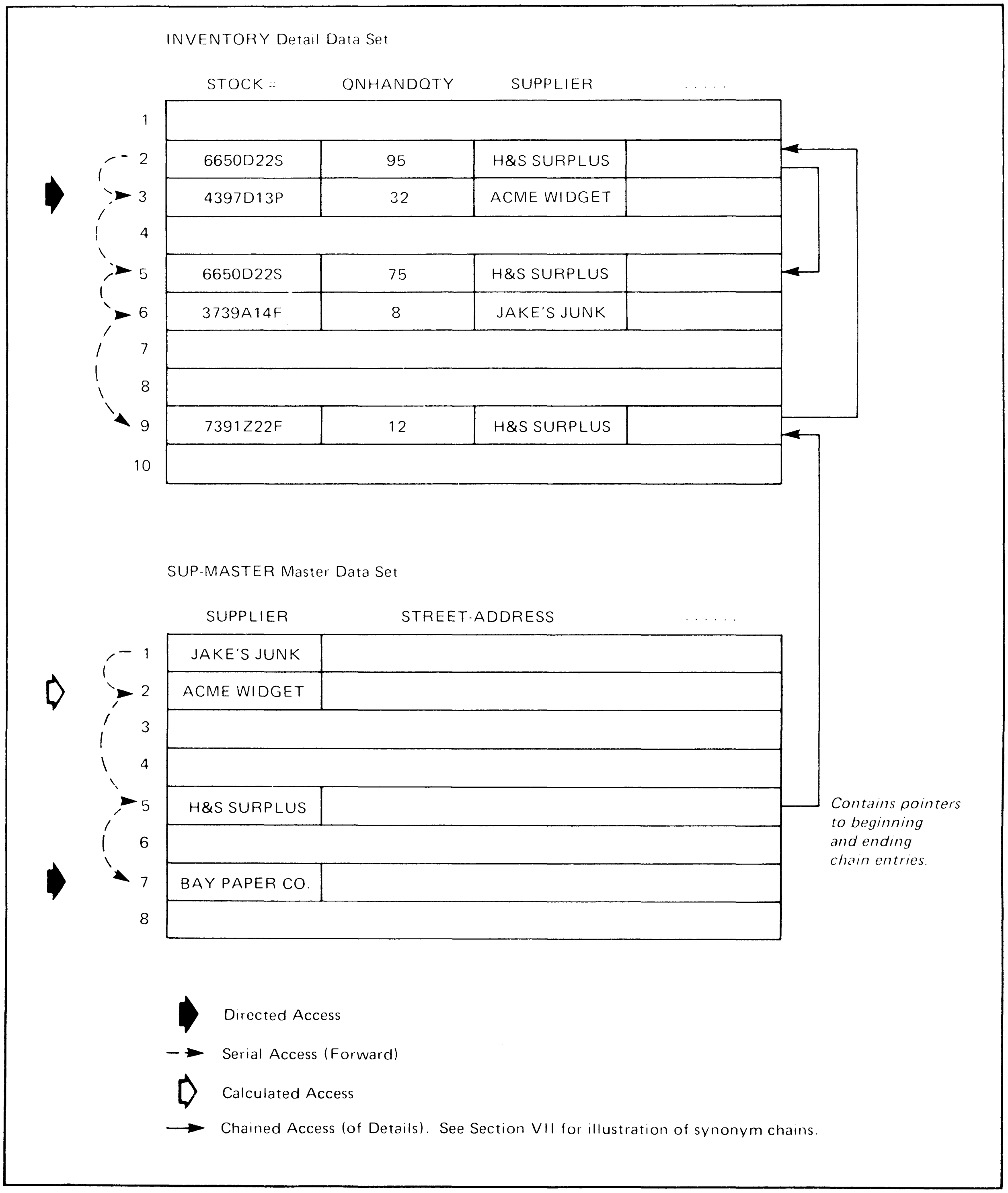
If an entry is read from a master data set, the chain pointers are synonym chain pointers and have no relationship to a path.

READING METHODS

The methods for requesting a data entry are categorized as

- directed access
- serial access
- calculated access
- chained access

All of these methods are available through the IMAGE library procedure DBGET. The chained access method also requires the use of the DBFIND procedure. Figure 4-2 illustrates the access methods using two data sets from the STORE data base.



INVENTORY Detail Data Set

	STOCK #	ONHANDQTY	SUPPLIER	.....
1				
2	6650D22S	95	H&S SURPLUS	
3	4397D13P	32	ACME WIDGET	
4				
5	6650D22S	75	H&S SURPLUS	
6	3739A14F	8	JAKE'S JUNK	
7				
8				
9	7391Z22F	12	H&S SURPLUS	
10				

SUP-MASTER Master Data Set

	SUPPLIER	STREET-ADDRESS	......
1	JAKE'S JUNK		
2	ACME WIDGET		
3			
4			
5	H&S SURPLUS		
6			
7	BAY PAPER CO.		
8			

Contains pointers to beginning and ending chain entries.
Directed Access
Serial Access (Forward)
Calculated Access
Chained Access (of Details). See Section VII for illustration of synonym chains.

Figure 4-2. Reading Access Methods (DBGET Procedure)

DIRECTED ACCESS

One method of selecting the data entry to be read is to specify its record number. This method is called directed access. If any entry exists at the record address specified by the calling program, IMAGE returns the values for the data items requested in the calling program's buffer. If no such entry exists, the program is notified by an exceptional condition return.

This access method can be used with any type of data set and is useful in situations where the calling program has already determined the record number of the entry to be read. For example, if a program surveys several entries using another access method to determine which one it wants to use in a report, it can save each record number and use the record number of the entry it selects to read the entry again using the directed access method.

If a program performs a directed read of record 3 of the INVENTORY data set, the entry marked with a solid black arrow in figure 4-2 is read. If a directed read of the SUP-MASTER data set record 7 is performed, the entry in that set marked with the same type of arrow is read.

LOCKING. If concurrent users are allowed to add to or delete from this data set, locking should be used during the search and report sequence to ensure the record numbers do not change before they are used. In this type of application, a data set lock is usually the most appropriate.

NOTE

When using this type of access with master data sets, you should be aware of migrating secondaries. These are described in Section VII.

SERIAL ACCESS

In this mode of retrieval, IMAGE starts at the most recently accessed storage location for the data set, called the *current record*, and sequentially examines adjacent records until the next entry is located. Data items from this entry are returned to the calling program, and its location becomes the current record.

You may use both forward and backward serial access. Forward serial access consists of retrieving the next greater-numbered entry and backward serial access consists of retrieving the previous lower-numbered entry. If no entry is located, IMAGE returns an exceptional condition, an end-of-file if the requested access is forward and a beginning-of-file if it is backwards.

Since there is no current record the first time a program requests an entry from a data set, a request for forward serial access causes IMAGE to search from record 1. Similarly, a backward serial retrieval begins at the highest numbered record.

The entries connected by a broken line in figure 4-2 are read by a program using the serial access method. If a forward serial read is performed on the INVENTORY data set before any other type of read, the entry in record number 2 is read. If another forward serial read is performed on the same data set, the entry in record 3 is read. On the other hand, if a serial read is performed and the current record is 6, the entry in record 9 is read. The next forward serial read returns an exceptional condition, end-of-file.

The serial access method can be used with any type of data set and is very useful if most or all of the data in the data set is to be retrieved, for example, to be used in a report. It is efficient to retrieve all the data serially, copy it to a file, and sort it with routines external to IMAGE before printing the report. The availability of serial access effectively allows you to use a data set in the same way you would use an MPE file. Thus, you have the advantages of IMAGE data base organization and the efficiency of serial access.

LOCKING. If concurrent users are allowed to modify the data set (access mode 1), you may wish to lock the data set or data base before you begin the serial access sequence. Locking will prevent entries from being added, modified, moved or removed by the other processes.

CALCULATED ACCESS

The calculated access method allows you to retrieve an entry from a master data set by specifying a particular search item value. For example, the SUP-MASTER data entry for the supplier Acme Widget shown in figure 4-2, can be retrieved with this method since SUPPLIER is a search item in the SUP-MASTER data set. IMAGE locates the entry in the data set whose search item value matches the requested value. The exact technique used to perform calculated access is described in Section VII.

Calculated access can be used only with master data sets. It is very useful for retrieving a single entry for some special purpose. For example, a program used infrequently to get information about a particular customer or supplier could use calculated access to quickly locate the information in the STORE data base.

CHAINED ACCESS

The chained access method is used to retrieve the next entry in the current chain. To perform chained access of detail data set entries, you must first locate the beginning of the chain you want to retrieve, and thus initiate the current chain, by calling the DBFIND procedure. The calling program specifies the name of the detail search item that defines the path to which the chain belongs and a value for the item. IMAGE determines which master set forms a path with the specified search item and locates the entry in that master data set whose search item value matches the specified value. The entry it locates contains pointers to the first and last entries in the desired chain and a count of the number of entries in the chain. This information is maintained internally and defines the *current path.*

If a program uses chained access to read the INVENTORY data set entries pertaining to the supplier H&S SURPLUS shown in figure 4-2, it must first call the DBFIND procedure to locate the chain head in the SUP-MASTER data set. The program specifies the INVENTORY data set, the SUPPLIER search item in the INVENTORY data set and the value H&S SURPLUS for that item. IMAGE uses a calculated read to locate the SUP-MASTER entry with a search item value of H&S SURPLUS. If the program then requests a forward chained read using the DBGET procedure, the entry in record 9 of INVENTORY, which is at the beginning of the chain, is read. If a backward chained read is requested, the entry in record 5 is read.

If the last call to DBGET used chained access to read the entry in record 9, the next forward chained read reads the entry in record 2 of the INVENTORY data set.

Once a current path, and chain, has been established for a data set, the calling program can use the chained access method of retrieving data. You may use both forward and backward chained access. In either case, if there are no more entries in the chain when you request the next one, DBGET returns an exceptional condition, beginning-of-chain or end-of-chain for backward and forward access, respectively.

Chained access to master data sets retrieves the next entry in the current *synonym* chain. The use of synonym chains applies to only a limited number of special situations. They are discussed in Section VII.

Chained access to detail data sets is particularly useful when you want to retrieve information about related events such as all inventory records for the H&S Surplus supplier in the STORE data base.

LOCKING. If concurrent users are allowed to modify data entries in the chain you are currently accessing, you may use locking to ensure data consistency. For example, suppose a chain consists of several data entries, each containing a line item from a particular order. If user A is performing a series of chained reads while user B is cancelling the order by deleting data entries one by one, user A may retrieve an incomplete order. To prevent this from happening, a lock may be established covering the group of data entries to be retrieved (the chain, in this case). This can usually be done with a single DBLOCK call. (Refer to the discussion of the locking facility later in this section.)

RE-READING THE CURRENT RECORD

The DBGET library procedure allows you to read the entry from the most recently accessed record again. You may want to do this in a program that has unlocked the data entry and locked it again and needs to check if the contents of the current entry have been changed.

Note that if a DBFIND procedure call has been made, the current record is zero and a request to re-read the entry causes DBGET to return a flag indicating that the current record contains no entry. See the DBGET procedure table 4-8 for more information.

UPDATING DATA

IMAGE allows you to change the values of data items that are not search or sort items if the user class number with which you opened the data base grants this capability to you. Before you call the DBUPDATE library procedure to change the item values, you must call DBGET to locate the entry you intend to update. This sets the current record address for the data set. The DBUPDATE library procedure uses the current record address to locate the data items whose values are to be changed.

A lock may be established before the call to DBGET to guard against accidental modification of the record by another user. This is recommended in any shared access mode (as discussed below).

When the program calls DBUPDATE it specifies the data set name, a list of data items to be changed, and the name of a buffer containing values for the items. For example, if a program changes the street address of a customer in the CUSTOMER data set of the STORE data base, the program can first locate the entry to be changed by calling DBGET in calculated access mode with the customer's account number and then calling the DBUPDATE procedure to change the value of the STREET-ADDRESS data item in that entry.

ACCESS MODES AND USER CLASS NUMBER

To update data items, the data base must be opened in access mode 1, 2, 3, or 4. If it is open in access mode 1, the data entry, data set, or data base must be locked while the update is happening.

IMAGE guarantees that all updates to a data entry will be carried out even if they are requested by different users simultaneously and locking is not used. To ensure this, IMAGE always completes the processing of one DBUPDATE request before it begins processing another. However, data consistency programs may still occur if an update is based on data values that are not current. For example, while withdrawing 10 items from the stock, two users may read the same data entry from the INVENTORY data set. If the current value of ONHANDQTY is 30 and they each subtract 10 from it and then update the entry, both updates will operate successfully but the new value will be 20 rather than 10. To prevent errors such as this, a lock covering the data entry can be put in effect before it is read and released after it is updated.

IMAGE attempts to enforce this locking technique for users in mode 1 by checking to see if an appropriate lock is in effect before executing an update. However, to have its proper effect, the lock should be made before the call to DBGET.

The password you use to open the data base must grant update capability to the data items you intend to change. The user class number associated with the password must either be in the write class list of the data set containing the items to be updated or in the read class list of the data set and in the write class list of the data item.

DELETING DATA ENTRIES

To delete an entry from a data set, you must first locate the entry to be deleted by reading it with the DBGET library procedure, or the DBFIND and DBGET procedures if it is advantageous to use chained access to locate the entry. You then call the DBDELETE procedure specifying the data set name. IMAGE verifies that your password and associated user class number allow you to delete the current entry of the specified data set.

If the detail data entry deleted is the only member of a detail chain linked to an automatic master, and all other chains linked to the same automatic master entry are empty, IMAGE automatically deletes the master entry.

If the data entry is a manual master data set, IMAGE verifies that the detail chains associated with the entry's search item, if any, are empty. If not, it returns an error condition to the calling program. For example, if a program attempts to delete the SUP-MASTER entry in figure 4-2 that contains a SUPPLIER value of H&S SURPLUS, an error condition is returned since a three-entry chain still exists in the INVENTORY detail data set.

To delete the CUSTOMER data set entry with ACCOUNT equal to 75757575, the program can call DBGET in calculated access mode specifying the CUSTOMER data set and the search item value 75757575. If the procedure executes successfully, the program then can call DBDELETE specifying the CUSTOMER data set to delete the current entry provided no chains in the related SALES detail data set contain search item values of 75757575.

ACCESS MODES AND USER CLASS NUMBERS

The data base must be opened with access mode 1, 3, or 4. If it is opened with access mode 1, the DBLOCK procedure must be used to lock the detail data entry, data set, or data base before an entry can be deleted and DBUNLOCK should be called after one or all desired entries have been deleted. As a general rule, the lock should be established before the whole delete sequence, in other words, before the call to DBGET that establishes which record is to be deleted. This will ensure that another user does not delete the data entry between the call to DBGET and the call to DBDELETE.

An entry cannot be deleted from a data set unless the user class number established when the data base is opened is in the data set write class list.

USING THE LOCKING FACILITY

The DBLOCK procedure applies a logical lock to a data base or one or more data sets or data entries. The DBUNLOCK procedure releases these locks.

Locking can be viewed as a means of communication and control to be used by mutually cooperating users. The locking facility provides a method for protecting the logical integrity of the data shared in a data base. With the DBLOCK procedure, application programs may isolate temporarily a subsection of the data base in order to perform a transaction against the isolated data. Locking is not required to protect the structure of the data base. IMAGE has internal mechanisms that do this.

If a program opens the data base in access mode 1 and locks a part of the data base, it can perform the transaction with the certain knowledge that no other user will modify the data until the application program issues a DBUNLOCK call. This is because IMAGE does not allow changes in access mode 1 unless a lock covers the data to be changed. If one process has the data base opened in access mode 1, IMAGE requires that all other processes that modify the data base must also operate in access mode 1.

The DBLOCK procedure operates in one of six modes. Modes 1 and 2 may be used for locking the data base and modes 3 and 4 for locking a data set. In modes 5 and 6, you describe the data base entity or entities to be locked using a *lock descriptor.*

At the data entry level, locking is performed on the basis of data item values. For example, suppose a customer requests a change in an order he has placed. The data entries for his account that are in the SALES data set may be locked while his order is changed and other data base activity may continue concurrently.

LOCK DESCRIPTORS

A lock descriptor is used to specify a group of data entries that are to be locked. It consists of a data set name or number, a data item name or number, a relational operator, and an associated value. For purposes of this discussion, the notation *dset* : *ditem relop value* is used. For example, the lock descriptor SALES: ACCOUNT = 89393899 requests locking of all the data entries in the SALES data set with an ACCOUNT data item equal to 89393899. Note that the result of specifying a single lock descriptor may be that none, one or many entries are locked depending on how many entries qualify.

The following relational operators may be used:

- less than or equal (< =)
- greater than or equal (> =)
- equal (= Δ or Δ =). Δ indicates a space character.

The value must be specified exactly as it is stored in the data base. A lock will succeed even if no data item with the specified value exists in the data set; no check is made to determine the existence of a particular data item value.

This allows you to use techniques such as issuing a lock to cover a data entry before you actually add it to the data set.

With the exception of compound items, any data item may be used in a lock descriptor. It need not be a search item.

IMAGE does not require that you have read or write access to a data set or data item in order to specify it in a lock request.

A process may specify any number of lock descriptors with a single DBLOCK call. For example, the following lock descriptors may be specified in one DBLOCK call:

```
CUSTOMER: ACCOUNT = 89393899
SALES: ACCOUNT = 89393899
SUP-MASTER: STATE = AZ
INVENTORY: ONHANDQTY < = 100
INVENTORY: ONHANDQTY > = 1500
```

Multiple calls to DBLOCK without intervening calls to DBUNLOCK are not allowed unless the program has Multiple RIN (MR) capability. (Refer to the discussion of multiple DBLOCK calls later in this section.)

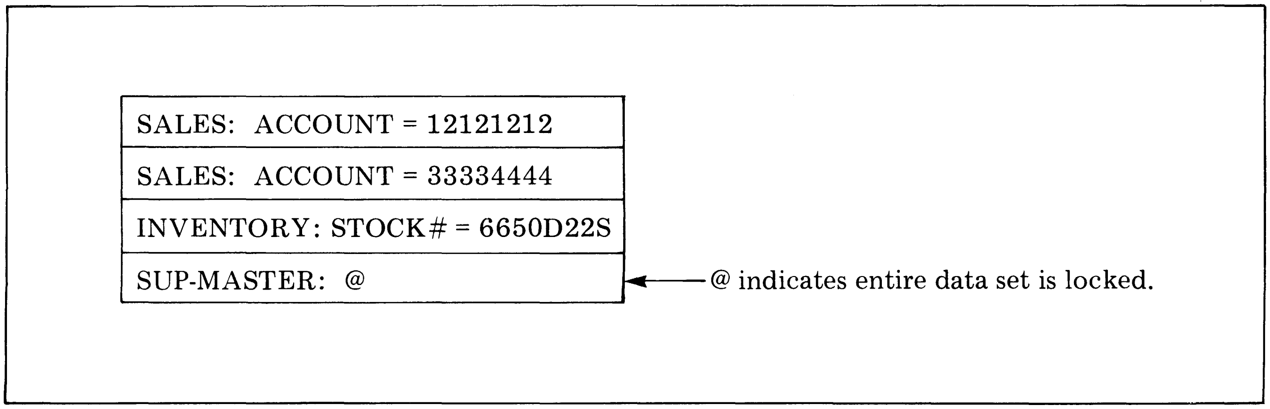
HOW LOCKING WORKS

The internal implementation of locking does not involve reading or writing to the data base element to be locked. IMAGE keeps a table of everything that is locked by all processes that have the data base opened. One table is associated with each data base. This table serves as a global list of lock descriptors. In locking mode 5 or 6, a data base lock is specified with the descriptor @: @ and a data set lock with *dset*: @. If you call DBLOCK in locking mode 1, 2, 3, or 4, IMAGE sets up the appropriate lock descriptor and puts it in the lock descriptor table.

Figure 4-2.1 illustrates the contents of this list in a situation where one process has locked all SALES data entries with ACCOUNT equal to 12121212 or equal to 33334444. Another process has locked all INVENTORY data entries with STOCK# equal to 6650D22S. A third process has locked the whole SUP-MASTER data set.

(Note that the figure illustrates what the table represents, not the actual internal format.)

When a lock request is made, IMAGE compares the newly specified lock descriptors with those that are currently in the list. If a conflict exists, IMAGE notifies the calling process that the entity cannot be locked or, if the process has requested unconditional locking, it is placed in a waiting state until the entity can be locked. If there are no conflicts, IMAGE adds the new lock descriptors to the list.



SALES: ACCOUNT = 12121212

SALES: ACCOUNT = 33334444

INVENTORY: STOCK# = 6650D22S

SUP-MASTER: @

Figure 4-2.1. Lock Descriptor List

CONDITIONAL AND UNCONDITIONAL LOCKING

You may request conditional or unconditional locking. If you request unconditional locking, IMAGE returns control to your calling program only after the specified entity has been locked. If you request conditional locking, IMAGE returns immediately. In this case, the condition code must be examined to determine whether or not the requested locks have been applied. If multiple lock descriptors are specified, the status area indicates the number that have been applied. The calling program should call DBUNLOCK if only a subset of the requested locks succeeded.

ACCESS MODES AND LOCKING

It is anticipated that access mode 1 will typically be used by applications implementing a locking scheme. In this mode, IMAGE enforces the following rules:

- To modify (DBPUT, DBDELETE, or DBUPDATE) a data entry, you must first issue a successful lock covering the affected data entry. It may be a data entry, data set, or data base lock.

- To add to or delete from (DBPUT or DBDELETE) a master data set, you must first successfully lock the data set or data base. To update (DBUPDATE) a master data set, data entry level locks are sufficient.

If your application opens the data base in access mode 2, it is recommended that you use locking to coordinate updates with other users.

IMAGE does not prevent any process from reading data even though another process holds a lock on it. If you want to ensure that no modifications are in progress while you are reading from the data base, you should place an appropriate lock on the data before starting. Therefore, you may want to use locking in access modes 2, 4, 5, and 6 to coordinate the reading and modifying sequences and ensure that they do not occur concurrently.

Since access mode 3 and 7 users have exclusive control of the data base and access mode 8 users allow concurrent reading only, locking need not be used in these modes.

AUTOMATIC MASTERS

When adding or deleting entries from a detail data set, you need not have locks covering the implicit additions or deletions that occur in any associated automatic masters.

LOCKING LEVELS

Locking can be viewed as operating on three levels: the whole data base, whole data sets, or data entries. IMAGE allows mixed levels of locking; for example, one user may be locking data entries and another locking the data set. In this situation, a request to lock the data set cannot succeed until all the currently locked data entries have been released. Subsequent requests to lock data entries, those that are made while the data set lock is pending, are placed in a queue behind the data set lock.

This principle is followed for data base locks also. If data set or data entry locks are in effect at the time a data base lock is requested, the data base lock must wait until they are released and all subsequent locking requests must wait behind the pending data base lock.

In either case, if the request is for a conditional lock, an exceptional condition is generated. (Refer to the DBLOCK condition word values, table 4-12.)

DECIDING ON A LOCKING STRATEGY

It is important, especially for on-line interactive applications, to establish a locking strategy at system design time. In general, locking is related to the *transaction*, the basic unit of work performed against a data base. Typically a transaction consists of several calls to IMAGE intrinsics to locate and modify data. For example, a transaction to add a new order with three line items may require several reads to locate customer information and several DBPUT calls to add the order detail records.

One characteristic of a transaction is that the data in the data base is consistent both before and after the transaction, but not while it is in progress. For example, a user reading the detail data set being modified by the above order transaction may only see some of the line items and may get no indication that the transaction is incomplete. This type of problem is referred to as "logical inconsistency" of data and can be prevented by using the locking facilities.

The general principle that should be applied for *any* transaction in a shared-access environment is:

- At the start of any transaction, establish locks that cover all data entries that you intend to modify (DBPUT, DBDELETE, or DBUPDATE) and/or all data entries which must not change during the transaction.

CHOOSING A LOCKING LEVEL

Because IMAGE needs more information to lock data entries than to lock the whole data base, program complexity tends to increase the lower the level of locks employed. Locking the whole data base or a single data set is the simplest operation, followed in increasing order of complexity by locking multiple data sets and locking data entries. At system design time, a compromise must be made between the benefits of low-level locking and the extra programming effort required.

Data entry locking should always give the best performance, however, there are situations in which the extra programming effort for data entry locking is not worthwhile. Some other considerations that may affect your choice of locking level are discussed below.

LOCKING AT THE SAME LEVEL. All programs concurrently accessing a data base should lock at the same level most of the time. For example, one process locking a data set will hold up all other processes that are attempting to lock entries in that set. Therefore, the attempt by the process locking at the data entry level to allow other processes to share the data base is nullified by the process locking at the data set level and the effect is as if all processes were locking at the data set level.

The rule of locking at the same level may be violated for infrequent operations such as exception handling or rare transactions.

LENGTH OF TRANSACTION. Generally, the longer the lock is to be held, the lower level it should be. In other words, if you are performing lengthy transactions (more than about 8 IMAGE calls), you should probably lock at the entry level. For transactions shorter than this, data base or data set locks will give approximately the same results.

An extreme case of a long transaction is one in which user dialog takes place while a lock is held. For example, a program may read some data entries, interact with a terminal operator, and modify some or all of the entries. A lock to cover this transaction may last several minutes which is an unacceptable amount of time to stop all data base or data set activity. In this situation, data entry level locking should be used.

Since the length of different transactions varies, the longest transaction (that is also frequently used) should guide the choice of locking level.

LOCKING DURING USER DIALOG. In the situation described above where a lock is held during interactive dialog with a terminal operator, the terminal timeout feature of MPE may be used to avoid having the locked entity inaccessible when the terminal operator is interrupted in the middle of the dialog. The timeout feature may be used to cause the terminal read to terminate automatically if no response is received within a certain time period. Refer to the discussion of FCONTROL in the *MPE Intrinsics Manual.*

CHOOSING A DATA ITEM FOR LOCKING

An important convention to follow in designing a locking scheme is that all programs sharing the data base concurrently use the same data item to lock data entries in a particular data set. At any point in time, IMAGE allows no more than one data item per data set to be used for locking purposes. However, several values of the data item may be locked at the same time. For example, if one process has successfully locked SALES: ACCOUNT = 54321000, another process may lock SALES: ACCOUNT = 11111111. If a request is made to unconditionally lock SALES: STOCK# = 8888X22R, the requesting process will be made to wait until all entries locked by ACCOUNT number are unlocked. Furthermore, any new requests for locking other SALES: ACCOUNT values will wait until SALES: STOCK# = 8888X22R is successfully locked and unlocked again.

With this in mind, it is apparent that it is more efficient if all processes locking data entries in the SALES data set use the same data item since it is much less likely that one process will have to wait until another process finishes using the data. Therefore, at system design time, decide which item will be used in each data set for lock specification purposes. (It may be useful to add comments in the schema indicating which item is the locking item for each set.)

EXAMPLES OF USING THE LOCKING FACILITY

The following examples list the order in which IMAGE intrinsics may be called when using the locking facility while performing various transactions. The examples refer to the STORE data base described in figures 2-5 and 2-6.

- *Add a new customer*

  1. DBLOCK the CUSTOMER data set or the whole data base.

  2. DBPUT new data entry in CUSTOMER data set.

  3. DBUNLOCK.

  The data set is locked because IMAGE requires a data set or data base lock to cover addition of an entry to a master data set.

- *Update an INVENTORY data entry to increase UNIT-COST for part 6650D22S by 12 percent*

  1. DBLOCK INVENTORY: STOCK# = 6650D22S. (Alternatively, the INVENTORY set or the whole data base can be locked.)

  2. DBFIND and DBGET the data entry that is locked in step 1.

  3. Compute new UNIT-COST = UNIT-COST + .12 * UNIT-COST.

  4. DBUPDATE the data entry that is locked.

  5. DBUNLOCK.

- *Insert a new product with a new supplier*

  1. DBLOCK the PRODUCT master data set.
  2. DBPUT new product data entry in PRODUCT master data set.
     (For example: 4444A33B CALIPER)
  3. DBUNLOCK.
  4. DBLOCK the SUP-MASTER data set.
  5. DBPUT new supplier data entry in SUP-MASTER data set.
  6. DBUNLOCK.
  7. DBLOCK INVENTORY: STOCK# = 4444A33B.
  8. DBPUT new data entry in INVENTORY data set for STOCK# = 4444A33B.
  9. DBUNLOCK.

  Note that this has been done as three transactions. If you did not want other users to see the data base with the supplier record present but with no inventory shown, you could perform one transaction with all the locks requested in one call to DBLOCK with mode 5 or 6.

- *Interactively modify an order for customer account 89393899*

  1. DBLOCK SALES: ACCOUNT = 89393899.
  2. DBFIND the CUSTOMER master data set entry with ACCOUNT = 89393899 in order to prepare to read the chain of SALES data entries with the same ACCOUNT value.
  3. DBGET each entry in the chain and display it to user until the correct order is located.
  4. Modify the contents of the data entry according to the users request.
  5. DBUNLOCK.

  All data entries for ACCOUNT 89393899 in the SALES data set are locked. Note that these locks are held while a dialog takes place with the terminal operator, therefore, the lock may be held for several minutes. For this type of transaction, it may be best to first perform a conditional lock to determine if the records are accessible. For example:

  1. DBLOCK SALES: ACCOUNT = 89393899 with mode 6.
  2. If the lock does not succeed, display a message such as:

     RECORDS BEING MODIFIED. WANT TO WAIT?

     If the response is NO then go to other processing.

     If the response is YES, call DBLOCK again with mode 5.

Table 4-2.1 contains guidelines that may be helpful in designing locking schemes for shared-access environments which include users who might modify the data base. Although data entry level locks are recommended in this table and illustrated in the examples above, data set or data base locks may be more appropriate for similar tasks depending upon other application requirements.

Table 4-2.1. Locking in Shared-access Environments

ACTION	RECOMMENDED LOCKS
Chained DBGET calls	Lock all data entries in the chain. This usually requires one lock descriptor.
Serial DBGET calls	Lock the data set.
Update a data entry (DBUPDATE)	Lock the data entry before calling DBGET to read the data entry. Unlock after the update.
Directed reads (DBGET calls)	These are not recommended in a shared environment. Lock the data set before determining which data entry is needed.
Add a data entry to a detail data set (DBPUT)	Any lock which covers this data entry, but preferably use the data item that was decided on as the 'lock item' for the data set.
Add to or delete from a master data set (DBPUT and DBDELETE)	Lock the data set or data base. This is mandatory if the data base is open in access mode 1.

ISSUING MULTIPLE CALLS TO DBLOCK

In order to guarantee that two processes cannot deadlock, once a call to DBLOCK is made by any process in a session or job, IMAGE does not allow a second call to be made unless the locks are cancelled with a call to DBUNLOCK first. There are two exceptions to this rule:

- A redundant call may be made to lock the whole data base with DBLOCK mode 1 or 2 provided the call relates to the same access path. The redundant call will have no effect. (This is allowed in order to maintain compatibility with earlier versions of IMAGE.)

- More than one DBLOCK call may be made if the program from which multiple DBLOCK calls are issued has Multiple RIN (MR) capability. (A user cannot prepare such a program unless he also has this capability. Refer to the *System Manager/System Supervisor Reference Manual* for more information.)

The DBLOCK procedure is similar to MPE global RIN locks* in that it may put a process into a waiting state and thus, may cause a deadlock to occur. For example, a deadlock may occur if process A is waiting for a global RIN to be freed by process B, and process B is waiting for a data base entity to be unlocked by process A. Therefore, issuing a DBLOCK in conjunction with a lock applied by MPE intrinsics such as LOCKGLORIN or FLOCK or by the COBOLLOCK procedure requires MR capability. (The use of MR capability is not recommended unless absolutely necessary.)

Users whose programs have MR capability and issue multiple DBLOCK calls are responsible for deadlock prevention. This type of locking must be done very carefully. Recovery from a deadlock requires a restart of the operating system.

*No RINs are actually involved. Refer to Appendix D for more information about MR capability.

No matter how many descriptors are listed in a single DBLOCK call, IMAGE guarantees that deadlocks will never occur provided that no executing program that accesses the data base has MR capability.

(Refer to appendix D for more information on the MR capability.)

RELEASING LOCKS

The locks held by a process for a particular access path of a data base are relinquished when the process calls DBUNLOCK, or automatically when the process closes the data base, terminates, aborts, or is aborted by an operator.

Failure of a program to release locks will result in other programs waiting indefinitely for any conflicting locks. These programs, while in a waiting state, cannot be aborted by the operating system. An attempt to abort such a waiting process will result in the abort taking effect as soon as the process obtains the lock for which it was waiting.

OBTAINING INFORMATION ABOUT THE DATA BASE STRUCTURE

The DBINFO library procedure allows you to acquire information programmatically about the data base. It provides information about data items, data sets, or data paths. The information returned is restricted by the user class number and access mode established when the data base is opened.

Any data items, data sets, or paths of the data base inaccessible to that user class or in that access mode are considered to be non-existent. For example, if the access mode grants only read access, this procedure will indicate that no data sets may have entries added.

The information that can be obtained through separate calls to DBINFO is summarized below.

In relation to data items, DBINFO can be used:

- to determine whether the user class number established when the data base is opened allows a specified data item value to be changed in at least one data set, or allows a data entry containing the item to be added or deleted.

- to get a description of a data item including the data item name, type, sub-item length, and sub-item count. This information corresponds to that which is specified in the item part of the schema.

- to determine the number of items in the data base available to the current user and to get a list of numbers identifying those items. The numbers indicate the position of each data item in the item part of the schema. The type of access, for example read-only, can also be determined.

- to determine the number of items in a particular data set available to the current user and get a list of those item numbers and the type of access available for each one.

In relation to data sets, DBINFO can be used:

- to determine whether the current user can add or delete entries to a particular data set.

- to get a data set description including the data set name, type, length in words and blocking factor for data entries in the set, number of entries in the set, and the capacity.

- to determine the number of data sets the current user can access and get a list of the data set numbers indicating the position of the data set definition in the set part of the schema. The type of access to each set is also indicated.

- to determine in which data sets a particular data item is available to the current user. The number of data sets, a list of data set numbers, and the type of access available for each set is returned.

In relation to paths, DBINFO can be used:

- to get information about the paths associated with a particular data set including the number of paths. If the data set is a master set, the information includes the data set number, search item number, and sort item number for each related detail. If the data set is a detail set, the information includes the master data set number of the related master data set, the detail search item number and sort item number for each path.

- to determine the search item number of a master data set or the search item number for the primary path of the detail and the data set number of the related master. In either case, if the search item is inaccessible to the current user, no information is returned.

SPECIAL USES OF DBINFO

If the application program uses data item and data set numbers when calling the other IMAGE procedures, it is good practice to determine these numbers by calling DBINFO at the beginning of the of the program to set up the numbers. It is not practical to code the numbers into the program since a change to the data base structure might require extensive changes to the application programs. Likewise it is inefficient and time consuming to call DBINFO throughout the program to determine these numbers. Many application programmers prefer the convenience and flexibility of using the data item and data set names in procedure calls.

DBINFO is useful when writing general inquiry applications similar to the QUERY data base inquiry facility.

CLOSING THE DATA BASE OR A DATA SET

After you have completed all the tasks you want to perform with the data base, you use the DBCLOSE library procedure to terminate access to it. When DBCLOSE is used for this purpose, all data set files and the root file are closed and the data segment containing the ULCB is released to the MPE system. If there are no other concurrent users of the data base, the extra data segment containing the DBCB is also released. All locks that you still have on the data base through the closed access path are automatically released.

The DBCLOSE procedure can also be used to terminate, temporarily or permanently, access to a data set. A data set can be closed or rewound. Rewinding consists of resetting the dynamic status information kept by IMAGE to its initial state. If a detail data set is closed or rewound, the current path does not change when the status information is initialized.

The purpose of closing a data set completely is to return the resources required by that data set to the MPE system without terminating access to the data base. A typical reason for rewinding a data set is to start at the first, or last, entry again when doing a forward or backward serial read.

It is important to close the data base before terminating a program to ensure that the files are closed in an orderly manner. This is particularly true when using IMAGE with programs operating under control of the BASIC Interpreter since termination of your BASIC program does not coincide with termination of the BASIC Interpreter process.

CHECKING THE STATUS OF A PROCEDURE

Each time a procedure is called, IMAGE returns status information in a buffer specified by the calling program and sets the *condition code* maintained by MPE in the status register. The condition code, or the condition word described later, should be checked immediately after IMAGE returns from the procedure to the calling program.

A condition code is always one of the following and has the general meaning shown:

Condition Code	General Meaning
CCE	The procedure performed successfully. No exceptional conditions were encountered.
CCG	An exceptional condition, other than an error, was encountered.
CCL	The procedure failed due to an invalid parameter or a system error.

The first word of the status information returned in the calling program's buffer is a *condition word* whose value corresponds to the condition code as follows:

Condition Code	Condition Word Value
CCE	0
CCG	>0
CCL	<0

The calling program must check either the condition code or the condition word to determine the success or failure of the procedure. The condition word is also used to indicate various exceptional conditions and errors. These are summarized in Appendix A.

The other words of status information vary with the outcome of the call and from one procedure to another. The content of these words is described in detail with each procedure definition later in this section and in Appendix A, which describes error conditions.

INTERPRETING ERRORS

IMAGE provides two library procedures, DBEXPLAIN and DBERROR, which you can use to interpret status information programmatically. DBEXPLAIN prints on the $STDLIST device an English language error message which includes the name of the data base and the name of the procedure that returned the status information. DBERROR performs a similar function but returns the information in a buffer specified by the calling program.

These procedures are intended primarily for use in debugging application programs rather than in interpreting errors in the production environment where more specific application messages are necessary.

ABNORMAL TERMINATION

Under certain conditions, the calling process may be terminated by IMAGE. Conditions giving rise to process termination and a description of the accompanying error messages are presented in Appendix A.

USING THE IMAGE LIBRARY PROCEDURES

The following pages contain the reference specifications for the IMAGE procedures, arranged alphabetically. The calling parameters for each procedure are defined in the order in which they appear in the call statement. Each parameter must be included when a call is made since their meaning is determined by their position.

Table 4-3 illustrates the forms of the call statements for the four languages that can be used to call the procedures. Section V contains examples of using the IMAGE procedures with these languages. It also provides a sample RPG program for those who want to use IMAGE with RPG.

Table 4-3. Calling an IMAGE Procedure

COBOL	CALL "*name*" USING *parameter,parameter , . . . , parameter*
FORTRAN	CALL *name* (*parameter,parameter , . . . , parameter*)
SPL	*name* (*parameter,parameter , . . . , parameter*)
BASIC	*linenumber* CALL *name* (*parameter,parameter , . . . , parameter*)

All procedures may be called directly from programs in any of the four host languages since they are not TYPE procedures, they do not use the SPL OPTION VARIABLE capability, and all parameters are call-by-reference word pointers.

INTRINSIC NUMBERS

The intrinsic number is provided for each procedure except DBEXPLAIN and DBERROR. This number which uniquely identifies the procedure within IMAGE and the MPE operating system, is returned with other status information when an error occurs. You can use it to identify the procedure that caused the error.

DATA BASE PROTECTION

When each procedure is called, IMAGE verifies that the requested operation is compatible with the user class number and access mode established when the data base is opened.

UNUSED PARAMETERS

When calling some procedures for a specific purpose, one of the parameters may be ignored, however, it must be listed in the call statement. An application program may find it useful to set up a variable named DUMMY to be listed as the unused parameter in these situations, as a reminder that the value of the parameter does not affect the procedure call.

THE STATUS ARRAY

When the status array is described in the following discussion, the contents reflect a successful execution of the procedure. If the procedure does not execute successfully, standard error information is returned in the status array. This information is described in Appendix A.

DBCLOSE

INTRINSIC NUMBER 403

Terminates access to a data base or terminates, temporarily or permanently, access to a data set, or rewinds a data set.

PARAMETERS

base is the name of an array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN. (Refer to the DBOPEN intrinsic reference specification later in this section for more information about the *base id.*)

dset is the name of an array containing the left-justified name of the data set to be closed or is an integer referencing the data set by number if *mode* equals 2 or 3. If *mode* equals 1, this parameter is ignored.

The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

mode is an integer indicating the type of termination desired.
If *mode* equals 1, access to the data base is terminated. Any locks through the closed access path are released.

If *mode* equals 2, the data set referenced by the *dset* array is closed, but locks held in the data set are not released.

If *mode* equals 3, the data set referenced by the *deset* array is rewound but not closed.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. The status array contents are:

Word	Contents
1	Condition word (see table 4-4)
2-4	Unchanged from previous procedure call using this array.
5-10	Procedure call information. Refer to Appendix A for a description of this information.

REWINDING AND CLOSING DATA SETS

Although a program should close the data base when it has finished using the data, it may never need to close or rewind a data set. This capability is provided to reset the dynamic status information for a user's access to a particular data set to its initial state, and to return system resources without terminating access to the data base. When a detail data set is rewound, the current path does not change.

Since *mode* 3 does not close and re-open the data set, it is more efficient than *mode* 2 if the data set is to be accessed immediately after it is rewound.

In either case, the first subsequent access to the data set is treated as if it were the first access since the data base was opened, except that setting of chain pointers is relative to the current path number at the time the call to DBCLOSE was made.

If a process has the same data base open multiple times, only the access path specified in *base* is closed.

If a process aborts either by itself due to, for example, a system violation or due to the console operator aborting it, a mode 1 DBCLOSE is automatically issued for all data bases opened by that process.

Table 4-4. DBCLOSE Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
-2	FCLOSE failure.
CALLING ERRORS:	
-11	Bad *base* parameter.
-21	Bad data set reference.
-31	Bad *mode.*
COMMUNICATIONS ERRORS:	
-101	DSCLOSE failure.
-102	DSWRITE failure.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
63	Bad DBCB.

Consult Appendix A for more information about these condition codes.

TEXT DISCUSSION:

Page 4-15

DBDELETE

INTRINSIC NUMBER 408

Deletes the current entry from a data set. The data base must be open in access mode 1, 3, or 4.

PARAMETERS

base is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

dset is the name of an array containing the left-justified name of the data set from which the entry is to be deleted or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or a blank

mode must be an integer equal to 1.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word (see table 4-5)
2	Zero
3-4	Unchanged current record address
5-6	Number of entries in chain. If master data set, the number is zero unless the deleted entry was a primary entry with synonyms. In this case, the number is one less than its previous value. If detail data set, the number is unchanged from the preceding procedure call.
7-10	Unchanged preceding and succeeding record numbers of a chain. If master data set and the new synonym chain count is greater than zero, the numbers reference the last and first synonym chain entries respectively.

MASTER DATA SETS

When deleting entries from master data sets, the following rules apply:

- 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.
- If the data base is open in access mode 1, a lock must be in effect on the data set or the whole data base.

Table 4-5. DBDELETE Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
−1	FOPEN intrinsic failure.
−3	FREADDIR failure.
−4	FREADLABEL failure.
CALLING ERRORS:	
−11	Bad *base* parameter.
−12	No lock covers the data entry to be deleted. (Occurs only if open in access mode 1.)
−14	Illegal intrinsic in current access mode.
−21	Bad data set reference.
−23	Data set not writable.
−31	Bad *mode*.
COMMUNICATIONS ERRORS:	
−102	DSWRITE failure.
−106	Remote data inconsistent.
−107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
17	No entry.
44	Chain head.
63	Bad DBCB.

Consult Appendix A for more information about these codes.

DETAIL DATA SETS

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 entry has been deleted, DBDELETE also deletes the master entry containing the chain heads.*

If the data base is open in access mode 1, you must establish a lock covering the data entry to be deleted before calling DBDELETE.

DATA SET INTERNAL INFORMATION

The current record number is unchanged. If a primary data entry with synonyms is deleted from a master data set and a secondary migrates, the backward and foward pointers reflect the new primary. In all other cases, the backward and forward pointers are unchanged when an entry is deleted. (Refer to Section VII for an explanation of migrating secondaries.)

TEXT DISCUSSION:

Page 4-12

*Note: In this case, the synonym chain information for the automatic master is set to zero. (See Section VII).

DBERROR

Moves an English language message, as an ASCII character string, to a buffer specified by the calling program. The message interprets the contents of the *status* array as set by a call to an IMAGE procedure.

PARAMETERS

status is the name of the array used as the *status* parameter in the IMAGE procedure call about which information is requested.

buffer is the name of an array in the calling program's data area, at least 36 words long, to which the message is returned.

length is an integer variable which is set by DBERROR to the positive byte length of the message placed in the *buffer* array. The length will never exceed 72 characters.

USING THE DBERROR MESSAGES

Like DBEXPLAIN, DBERROR messages are intended and appropriate for use while debugging application programs. The errors they describe are, for the most part, errors that do not occur in a debugged and running program.

Some errors or exceptional conditions are expected to occur, even in a production environment. For example, the MPE intrinsic DBOPEN may fail due to concurrent data base access. In this case, printing the DBERROR message:

DATA BASE OPEN EXCLUSIVELY

may be perfectly acceptable, even to the person using the application program. However, in many cases a specific message produced by the application program is preferable to the one produced by DBERROR. A DBFIND error generated by the application program, such as:

THERE ARE NO ORDERS FOR THAT PART NUMBER

would be more meaningful to a user entering data at a terminal than the DBERROR message:

THERE IS NO CHAIN FOR THE SPECIFIED SEARCH ITEM VALUE

Table 4-6 lists all messages that can be returned by DBERROR with their corresponding condition word values. Several messages may correspond to one condition word and the interpretation of the code depends on the context in which it is returned. Variable information is represented by a lowercase word or phrase.

Table 4-6. DBERROR Messages

CONDITION WORD	DBERROR MESSAGE
0	SUCCESSFUL EXECUTION – NO ERROR
-1	NO SUCH DATA BASE DATA BASE OPEN IN AN INCOMPATIBLE MODE BAD ACCOUNT REFERENCE BAD GROUP REFERENCE BAD ROOT FILE REFERENCE VIRTUAL MEMORY NOT SUFFICIENT TO OPEN ROOT FILE DATA BASE ALREADY OPEN FOR MORE THAN READ DATA BASE IN USE DATA BASE OPEN EXCLUSIVELY MPE SECURITY VIOLATION MPE FILE ERROR decimal integer RETURNED BY FOPEN ON { ROOT FILE DATA SET # decimal integer }
-2	MPE FILE ERROR decimal integer RETURNED BY FCLOSE ON { ROOT FILE DATA SET # decimal integer }
-3	MPE FILE ERROR decimal integer RETURNED BY FREADDIR ON { ROOT FILE DATA SET # decimal integer }
-4	MPE FILE ERROR decimal integer RETURNED BY FREADLABEL ON { ROOT FILE DATA SET # decimal integer }
-9	MPE ERROR %octal integer RETURNED BY GETDSEG OF decimal integer WORDS
-11	BAD DATA BASE NAME OR PRECEDING BLANKS MISSING BAD DATA BASE REFERENCE (FIRST 2 CHARACTERS)
-12	IMAGE procedure name CALLED WITHOUT COVERING LOCK IN EFFECT
-14	CALLS TO IMAGE procedure name NOT ALLOWED IN ACCESS MODE decimal integer
-21	BAD PASSWORD – GRANTS ACCESS TO NOTHING DATA ITEM NONEXISTENT OR INACCESSIBLE SPECIFIED SET IS NOT AN ACCESSIBLE DETAIL DATA SET DATA SET NONEXISTENT OR INACCESSIBLE
-23	USER (CLASS) LACKS WRITE ACCESS TO DATA SET
-24	DBPUT NOT ALLOWED ON AUTOMATIC MASTER DATA SET

Table 4-6. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE
-31	DBGET MODE **decimal integer** ILLEGAL FOR DETAIL DATA SET DBGET MODE **decimal integer** BAD — SPECIFIED DATA SET LACKS CHAINS BAD (UNRECOGNIZED) **IMAGE procedure name** MODE: **decimal integer**
-32	UNOBTAINABLE ACCESS MODE: AOPTIONS REQUESTED: %**octal integer,** GRANTED: %**octal integer**
-51	LIST TOO LONG OR NOT PROPERLY TERMINATED
-52	ITEM SPECIFIED IS NOT AN ACCESSIBLE SEARCH ITEM IN THE SPECIFIED SET BAD LIST — CONTAINS ILLEGAL OR DUPLICATED DATA ITEM REFERENCE
-53	DBPUT LIST IS MISSING A SEARCH OR SORT ITEM
-60	ILLEGAL FILE EQUATION ON ROOT FILE
-91	ROOT FILE (DATA BASE) NOT COMPATIBLE WITH CURRENT IMAGE INTRINSICS
-92	DATA BASE REQUIRES CREATION (VIRGIN ROOT FILE)
-94	DATA BASE BAD — MAY NOT BE ACCESSED IN MODE decimal integer
-100	MPE ERROR decimal integer RETURNED BY DSOPEN
-101	MPE ERROR decimal integer RETURNED BY DSCLOSE
-102	MPE ERROR decimal integer RETURNED BY DSWRITE
-103	REMOTE 3000 STACK SPACE INSUFFICIENT
-104	REMOTE 3000 DOES NOT SUPPORT IMAGE
-105	REMOTE 3000 MPE ERROR %**octal integer** RETURNED BY GETDSEG OF decimal integer WORDS
-106	REMOTE 3000 DATA INCONSISTENT
-107	DS/3000 SYSTEM ERROR
-120	INSUFFICIENT STACK SPACE FOR DBLOCK
-121	ILLEGAL LOCK DESCRIPTOR COUNT
-122	BOUNDS VIOLATION ON DESCRIPTOR LIST
-123	ILLEGAL RELATIONAL OPERATOR
-124	DESCRIPTOR LENGTH ERROR; MUST BE 9 OR MORE
-125	ILLEGAL SET NAME OR NUMBER IN DESCRIPTOR
-126	ILLEGAL ITEM NAME OR NUMBER IN DESCRIPTOR
-127	ILLEGAL ATTEMPT TO LOCK ON A COMPOUND ITEM
-128	VALUE FIELD TOO SHORT FOR THE ITEM SPECIFIED
-129	P28 IS LONGEST P-TYPE ITEM THAT CAN BE LOCKED
-130	ILLEGAL DECIMAL DIGIT IN TYPE ‘P’ DATA VALUE
-131	LOWERCASE CHARACTER IN TYPE ‘U’ DATA VALUE
-132	ILLEGAL DIGIT IN TYPE ‘Z’ DATA VALUE
-133	ILLEGAL SIGN CHARACTER IN TYPE ‘Z’ DATA VALUE
-134	TWO LOCK DESCRIPTORS CONFLICT IN SAME REQUEST
-135	DBLOCK CALLED WITH LOCKS ALREADY IN EFFECT IN THIS JOB/SESSION
-136	DESCRIPTOR LIST LENGTH EXCEEDS 2047 WORDS
10	BEGINNING OF FILE
11	END OF FILE
12	DIRECTED BEGINNING OF FILE

Table 4-6. DBERROR Messages (Continued)

CONDITION WORD	DBERROR MESSAGE	
13	DIRECTED END OF FILE	
14	BEGINNING OF CHAIN	
15	END OF CHAIN	
16	THE DATA SET IS FULL	
17	THERE IS NO CHAIN FOR THE SPECIFIED SEARCH ITEM VALUE THERE IS NO ENTRY WITH THE SPECIFIED KEY VALUE THERE IS NO PRIMARY SYNONYM FOR THE SPECIFIED KEY VALUE NO CURRENT RECORD OR THE CURRENT RECORD IS EMPTY (CONTAINS NO ENTRY) THE SELECTED RECORD IS EMPTY (CONTAINS NO ENTRY)	
18	BROKEN CHAIN – FORWARD AND BACKWARD POINTERS NOT CONSISTENT	
20	DATA BASE CURRENTLY LOCKED BY ANOTHER USER SETS OR ENTRIES LOCKED WITHIN DATA BASE	
22	DATA SET ALREADY LOCKED	Conditional Locks Only
23	CANNOT LOCK SET DUE TO LOCKED ENTRIES WITHIN IT	
24	ENTRIES CURRENTLY LOCKED USING DIFFERENT ITEM	
25	CONFLICTING ENTRY LOCK ALREADY IN EFFECT	
41	DBUPDATE WILL NOT ALTER A SEARCH OR SORT ITEM	
42	DBUPDATE WILL NOT ALTER A READ-ONLY DATA ITEM	
43	DUPLICATE KEY VALUE IN MASTER	
44	CAN'T DELETE A MASTER ENTRY WITH NON-EMPTY DETAIL CHAINS	
50	USER'S BUFFER IS TOO SMALL FOR REQUESTED DATA	
61	PROCESS HAS THE DATA BASE OPEN 63 TIMES; NO MORE ALLOWED	
62	IMAGE DATA BASE CONTROL BLOCK FULL	
63	DATA BASE CONTROL BLOCK DISABLED; POTENTIAL DAMAGE; ONLY DBCLOSE ALLOWED	
64	NO ROOM FOR DBCB ENTRY IN PCBX (MPE PORTION OF STACK)	
66	DBCB POINTED TO BY ROOT FILE DOES NOT MATCH	
1xx	THERE IS NO CHAIN HEAD (MASTER ENTRY) FOR PATH **decimal integer: xx**	
2xx	THE CHAIN FOR PATH **decimal integer: xx** IS FULL (CONTAINS 65535 ENTRIES)	
3xx	THE AUTOMATIC MASTER FOR PATH **decimal integer:** xx IS FULL	
Others	UNRECOGNIZED CONDITION WORD: **decimal integer**	

TEXT DISCUSSION:

Page 4-16

DBEXPLAIN

Prints a multi-line message on the $STDLIST device which describes an IMAGE procedure call and explains the call's results as recorded in the calling program's *status* array.

PARAMETERS

status	is the name of the array used as the *status* parameter in the IMAGE procedure call about which information is requested.

NOTE

The *base, qualifier, dset,* and *password* parameters, if required for the procedure which put the results in the status area, must be unchanged when the call is made to DBEXPLAIN since information is taken from them as well.

MESSAGE FORMAT

Table 4-7 contains the general format for lines 2 through 6 of the message which is sent to $STDLIST. Elements surrounded by brackets are sometimes omitted. Braces indicate that only one of the choices shown will be printed. Lines 5 and 6 are printed only if, during the preparation of lines 2, 3, and 4, IMAGE detects that the status array contents are invalid, unrecognizable or incomplete, or if a message must be truncated to fit on a single line.

If the *status* array contents appear to be the result of something other than an IMAGE procedure call or if the array is used by the called procedure for information other than that discussed here, the second choice for line 3 is printed. This would be the case for a successful call to DBGET which uses all ten *status* words to return a condition word, lengths, and record numbers.

If the *status* array contains an unrecognized error code, the second line 4 choice is printed.

If the condition word is greater than or equal to zero, the word "ERROR" in line 2 is replaced by "RESULT" because non-negative condition words indicate success or exceptional conditions such as end-of-chain. Condition word values are explained in Appendix A.

You can use the *offset* information to locate the specific call statement that generated the *status* array contents if the call is made with a programming language which enables you to determine displacements of program statements or labels within the code. The identity of the code segment is not printed because it cannot be determined by DBEXPLAIN. Therefore, you need to be familiar with the program's functioning in order to locate the correct call. The offset portion of line 2 is printed only if the *status* array appears to be set by an IMAGE library procedure call and contains valid offset information.

Table 4-7. DBEXPLAIN Message Format

LINE	FORMAT
1	(a blank line)
2	$\text{IMAGE}\left\{\begin{matrix}\text{ERROR}\\ \text{RESULT}\end{matrix}\right\}$ [AT offset]: CONDITION WORD = conword
3	$\left\{\begin{matrix}\text{intrinsicname, MODE x, ON [setname OF] basename [;PASSWORD=password]}\\ \text{IMAGE CALL INFORMATION NOT AVAILABLE}\end{matrix}\right\}$
4	$\left\{\begin{matrix}\text{message}\\ \text{UNRECOGNIZED CONDITION WORD: conword}\end{matrix}\right\}$
5	[OCTAL DUMP OF STATUS ARRAY FOLLOWS]
6	[octal display]
7	(a blank line)

where:

offset is the octal PB-relative offset within the user's code segment of the IMAGE procedure call. See the **MPE Intrinsics Reference Manual** for a discussion of PB (program base) relative addresses.

conword is the condition word (from the first word of *status*) printed as a decimal integer and corresponding to the condition words described in Appendix A.

intrinsicname is the name of the IMAGE library procedure (intrinsic) which was called and which set the contents of the *status* array.

x is the value of the *mode* parameter printed as a decimal integer.

setname is the value of the second parameter, usually a data set name or number, as passed to the procedure which set the *status* array contents. The second parameter can be a data item name or number if the procedure in question is DBINFO. If the procedure is DBOPEN, DBLOCK, DBUNLOCK, or certain modes of DBINFO or DBCLOSE, setname is omitted.

password is printed at the end of line 3 only if the error relates to the *password* parameter of DBOPEN.

basename is the data base name specified in the procedure which was called and set the *status* array contents.

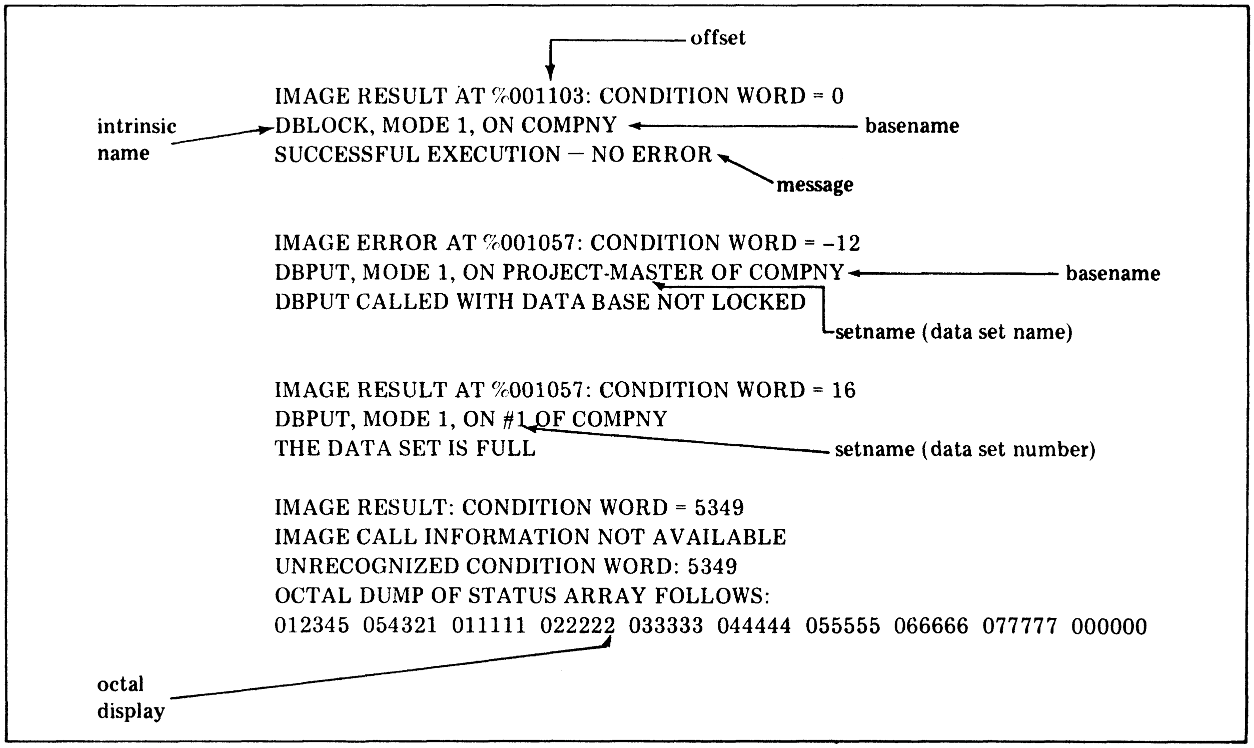
message is an English language description of the result based on the condition word and other *status* array information. The message is generated by the DBERROR procedure which is also described in this section. See Table 4-17 for all possible line 4 messages.

octal display is a listing of each word of *status* printed as a string of 6 octal digits. Adjacent *status* words are separated by a blank and the entire line is 69 characters long.

DBEXPLAIN

EXAMPLE

Figure 4-3 contains four examples of messages generated by DBEXPLAIN.



```
IMAGE RESULT AT %001103: CONDITION WORD = 0
DBLOCK, MODE 1, ON COMPNY
SUCCESSFUL EXECUTION — NO ERROR

IMAGE ERROR AT %001057: CONDITION WORD = -12
DBPUT, MODE 1, ON PROJECT-MASTER OF COMPNY
DBPUT CALLED WITH DATA BASE NOT LOCKED

IMAGE RESULT AT %001057: CONDITION WORD = 16
DBPUT, MODE 1, ON #1 OF COMPNY
THE DATA SET IS FULL

IMAGE RESULT: CONDITION WORD = 5349
IMAGE CALL INFORMATION NOT AVAILABLE
UNRECOGNIZED CONDITION WORD: 5349
OCTAL DUMP OF STATUS ARRAY FOLLOWS:
012345 054321 011111 022222 033333 044444 055555 066666 077777 000000
```

Figure 4-3. Sample DBEXPLAIN Messages

TEXT DISCUSSION:
Page 4-16

DBFIND

INTRINSIC NUMBER 404

Locates master set entry that matches the specified search item value and sets up pointers to the first and last entries of a detail data set chain in preparation for chained access to the data entries which are members of the chain. The path is determined and the chain pointers located on the basis of a specified search item and its value.

PARAMETERS

base is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

dset is the name of an array containing the left-justified name of the detail data set to be accessed or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

mode must be an integer equal to 1.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word (see table 4-8)
2	Zero
3-4	Doubleword current record number set to zero
5-6	Doubleword count of number of entries in chain
7-8	Doubleword record number of last entry in chain
9-10	Doubleword record number of first entry in chain

item is the name of an array containing a left-justified name of the detail data set search item or is an integer referencing the search item number that defines the path containing the desired chain. The name may be 16 characters long or, if shorter, terminated by a semicolon or blank. The specified search item defines the path to which the chain belongs.

argument contains a value for the search item to be used in calculated access to locate the desired chain head in the master data set.

DATA SET INTERNAL INFORMATION

The current values of chain count, backward pointer, and forward pointer for the detail data set referenced in *dset* are replaced by the corresponding value from the chain head. A current path number, which is maintained internally, is set to the new path number and the current record number for the data set is set to zero. Refer to Section VII for further information about chain heads and internally maintained data set information.

Note that although a master set entry exists with the specified search item value, the data set chain may be empty.

Table 4-8. DBFIND Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
–1	FOPEN intrinsic failure.
–3	FREADDIR failure.
–4	FREADLABEL failure.
CALLING ERRORS:	
–11	Bad *base* parameter.
–21	Bad data set reference.
–31	Bad *mode.*
–52	Bad *item.*
COMMUNICATIONS ERRORS:	
-102	DSWRITE failure.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
17	No master entry
63	Bad DBCB.

Consult Appendix A for more information on these conditions.

TEXT DISCUSSION:
Page 4-8, 4-11

DBGET

INTRINSIC NUMBER 405

Provides eight different methods for reading the data items of a specified entry.

PARAMETERS

base is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

dset is the name of an array containing the left-justified name of the data set to be read or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

mode contains an integer between 1 and 8, inclusive, which indicates the reading method. The methods are:

Mode	Method
1 (Re-read)	Read the entry at the internally maintained current record address. *argument* parameter is ignored.
2 (Serial Read)	Read the first entry whose record address is greater than the internally maintained current address. *argument* parameter is ignored.
3 (Backward Serial Read)	Read the first entry whose record address is less than the internally maintained current address. *argument* parameter is ignored.
4 (Directed Read)	Read the entry, if it exists, at the record address specified in the *argument* parameter. *argument* is treated as a doubleword record number.
5 (Chained Read)	Read the next entry in the current chain, the entry referenced by the internally maintained forward pointer. *argument* parameter is ignored.
6 (Backward Chained Read)	Read the previous entry in the current chain, the entry referenced by the internally maintained backward pointer. *argument* is ignored.
7 (Calculated Read)	Read the entry with a search item value that matches the value specified in *argument*. The entry is in the master data set specified by *dset*.
8 (Primary Calculated Read)	Read the entry occupying the primary address of a synonym chain using the search item value specified in argument to locate the entry. If the entry is not a primary entry in a master data set specified by *dset*, it is not read. (See Section VII for synonym chain description.)

status is the name of a ten-word array in which IMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word (see table 4-9)
2	Integer word length of the logical entry read into the *buffer* array.
3-4	Doubleword record number of the data entry read.
5-6	Doubleword zero, unless the entry read is a primary entry in which case it is the number of entries in the synonym chain.
7-8	Doubleword record number of the preceding entry in the chain of the current path.
9-10	Doubleword record number of the next entry in the chain of the current path.

list is the name of an array containing an ordered set of data item identifiers, either names or numbers. The values for these data items are placed in the array specified by the *buffer* parameter in the same order as they appear in the *list* array.

The *list* array may contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No embedded blanks are allowed and no name may appear more than once.

When referencing by number, the first word of the list array is an integer *n* which is followed by *n* unique data item numbers (one-word integers).

The *list* not only specifies the data items to be retrieved immediately but is saved internally by IMAGE as the *current list* for this data set. The *current list* is unchanged until a different list is specified in a subsequent call to DBGET, DBPUT, or DBUPDATE for the same access path and data set.

Some special list constructs are allowed. These are described in table 4-14 with the DBPUT procedure. List processing is a relatively high overhead operation which may be shortened in subsequent calls by using the asterisk construct to specify that the *current list* is to be used. Use of this construct can save considerable processing time.

buffer is the name of the array to which the values of data items specified in the *list* array are moved. The values are placed in the same order as specified in the *list* array. The number of words occupied by each value corresponds to the number required for each data type multiplied by the sub-item count.

argument is ignored except when *mode* equals 4, 7, or 8.

If *mode* is 4, *argument* contains a doubleword record number of the entry to be read.

If *mode* is 7 or 8, *argument* contains a search item value for the master data set referenced by *dset*.

Table 4-9. DBGET Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
-1	FOPEN intrinsic failure.
-3	FREADDIR failure.
-4	FREADLABEL failure.
CALLING ERRORS:	
-11	Bad *base* parameter.
-21	Bad data set reference.
-31	Bad *mode.*
-51	Bad *list* length.
-52	Bad *list* or bad *item.*
COMMUNICATIONS ERRORS:	
-102	DSWRITE Failure.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
10	Beginning of file. (mode 3)
11	End of file. (mode 2)
12	Directed beginning of file. (mode 4)
13	Directed end of file (mode 4)
14	Beginning of chain. (mode 6)
15	End of chain. (mode 5)
17	No entry. (modes 1, 4, 7, 8)
18	Broken chain. (modes 5 or 6)
50	Buffer too small.
62	DBCB full.
63	Bad DBCB.

Consult Appendix A for more information about these conditions.

DATA SET INTERNAL INFORMATION

The internal backward and forward pointers for the data set are replaced by the current path's chain pointers from the entry just read. If the data set is a master, they are synonym chain pointers (see Section VII). If it is a detail with at least one path, the current path is the one established by the last successful call to DBFIND, or if no call has been made it is the primary path. If there are no paths defined, the internal pointers are set to zeroes.

The location of the entry just read becomes the current record for the data set.

TEXT DISCUSSION:

Page 4-10

DBINFO

INTRINSIC NUMBER 402

Provides information about the data base being accessed. The information returned is restricted by the user class number established when the data base is opened; any data items, data sets, or paths of the data base which are inaccessible to that user class are considered to be non-existent.

PARAMETERS

base is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* is returned by DBOPEN.

qualifier is the name of an array containing a data set or data item name or is an integer referencing a data item or data set depending on the value of the *mode* parameter. The relationship of *mode* and *qualifier* is explained in table 4-10. The form of this parameter is identical to the *dset* and *item* parameters described with the DBPUT and DBFIND procedures.

mode is an integer indicating which type of information is desired. The available *modes* and information supplied with each are described in table 4-10. (data item modes 1*nn*,data set modes 2*nn*, path modes 3*nn*)

status is the name of a ten-word array in which IMAGE returns status information about the procedure. The status array contents are:

Word	Contents
1	Condition word (see table 4-11)
2	Word length of information in *buffer* array
3-4	Unchanged from previous procedure call using this array
5-10	Information about the procedure call and its results. Refer to Appendix A for a description of this information.

buffer is the name of an array in which the requested information is returned. The contents of the *buffer* array vary according to the *mode* parameter used. They are also described in table 4-10.

Table 4-10. *mode* and *qualifier* Values and Results

mode	PURPOSE	*qualifier*	*buffer* ARRAY CONTENTS	COMMENTS
101	Defines type of access available for specific item.	data item name or number	word 1 ± data item number	If negative, data item can be updated or entry containing it can be added or deleted in at least one data set
102	Describes specific data item.	data item name or number	word 1 . . 8 data item name	Left-justified and padded with blanks, if necessary.
			9 data type Δ	(I,J,K,R,U,X,Z,P) Δ indicates blank
			10 sub-item length 11 sub-item count 12 0 13 0	integers
103	Identifies all data items available in data base and type of access allowed.	(ignored)	word 1 n	n = number of data items available
			2 ± data item number . . . . . n+1 ± data item number	Arranged in data item number order. If positive, read-only access. If negative, update or modify access in at least one data set.
104	Identifies all data items available in specific data set and type of access allowed.	data set name or number	(Same as *mode* 103)	(Same as *mode* 103 *except* arranged in order of occurrence in data entry.)
201	Defines type of access available for specific data set.	data set name or number	word 1 ± data set number	If negative, entries can be added or deleted.
202	Describes specific data set	data set name or number	word 1 . . 8 data set name	Left-justified and padded with blanks, if necessary.
			9 set type Δ	(M,A,D) Δ indicates blank
			10 entry word-length 11 blocking factor 12 0 13 0	integers
			14 15 number of entries in set 16 17 capacity of set	doubleword integers

Table 4-10. *mode* and *qualifier* Values and Results (Continued)

mode	PURPOSE	*qualifier*	*buffer* ARRAY CONTENTS	COMMENTS
203	Identifies all data sets available in data base and type of access allowed.	(ignored)	word 1 n 2 ± data set number . . . . . . n+1 ± data set number	n = number of data sets available Arranged in data set number order. If positive, read and possibly data item update access. If negative, modify access allowed.
204	Identifies all data sets available which contain specified data item and type of access allowed.	data item name or number	(Same as *mode* 203)	(Same as *mode* 203)
301	Identifies paths defined for specified data set.	data set name or number	word 1 n 2 data set number 3 search item number 4 sort item number . . . . 3n−1 data set number 3n search item number 3n+1 sort item number Note: If sort item is zero, none exists or it is inaccessible. A path designator is not included if user does not have access to search item.	n = number of paths 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. Path designators presented in order of their appearance in schema.
302	Identifies search item for specified data set.	master data set name or number	word 1 search item number 2 0	In master set, zero if inaccessible
		OR		
		detail data set name	word 1 search item number 2 data set number	In detail set. For primary path. Of related master. Both are zero if search item is inaccessible.

Table 4-11. DBINFO Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
-1	FOPEN intrinsic failure.
-4	FREADLABEL failure.
CALLING ERRORS:	
-11	Bad *base* parameter.
-21	Bad data item reference.
-31	Bad *mode.*
COMMUNICATIONS ERRORS:	
-102	DSWRITE failure.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
50	Buffer too small.
63	Bad DBCB.

Appendix A contains more information about these conditions.

TEXT DISCUSSION:

Page 4-14

DBLOCK

INTRINSIC NUMBER 409

Applies a logical lock to a data base, data set, or one or more data entries.

PARAMETERS

base is the name of the array used for the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

qualifier is the name of an integer referencing the data set number or an array containing a data set name or the set of lock descriptors. The format for lock descriptors is given below.

mode contains an integer indicating the type of locking desired.

Locking Mode — Type of Locking

1 (Data base, unconditional) DBLOCK applies an unconditional lock to the whole data base, returning to the calling program only after the lock is successful (or if an error occurs). The *qualifier* parameter is ignored.

2 (Data base, conditional) DBLOCK applies a conditional lock to the data base and returns immediately. A condition word of zero indicates success. A non-zero condition word indicates the reason for failure. (Refer to table 4-12.)

3 (Data set, unconditional) DBLOCK applies an unconditional lock to a data set. The *qualifier* parameter must specify the name of an array containing the left-justified name of the data set or the name of an integer referencing the data set number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

The data set need not be accessible for read or write access to the user requesting the lock.

4 (Data set, conditional) DBLOCK applies a conditional lock of the same type as mode 3. It always returns to the calling program immediately. A condition word of zero indicates success and a non-zero condition word indicates the reason for failure. (Refer to table 4-12.)

5 (Data entries, unconditional) DBLOCK applies unconditional locks to the data entries specified by lock descriptors. The *qualifier* parameter must specify the name of an array containing the lock descriptors. The format of the array is shown in figure 4-4. It returns only when all the locks have been acquired.

Locking Mode	Type of Locking
6 (Data entries, conditional)	DBLOCK applies conditional locks of the same type as mode 5. If multiple lock descriptors are specified, a return is made when DBLOCK encounters a lock descriptor that it cannot apply. All locks that have been applied until that point are retained. Since the locks are not executed in the order supplied by the user, it is not predictable which locks are held and which are not after an unsuccessful mode 6 DBLOCK. Status word 2 indicates how many lock descriptors were actually successful. It is recommended that a DBUNLOCK be issued after any unsuccessful mode 6 DBLOCK.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. The status array contents are:

Word	Contents
1	Condition word (see table 4-12)
2	The number of lock descriptors that were successfully applied in the DBLOCK request. For successful locks in modes 1 through 4 this will be 1.
3	If condition word = 20, this word contains 0 if data base locked, 1 if data set or entries locked.
4	Reserved: Contents undefined.
5-10	Information about the procedure call and its results. Refer to Appendix A for a complete description.

LOCK DESCRIPTOR ARRAY FORMAT

The format of the array containing a list of lock descriptors is illustrated in figure 4-4. The number of lock descriptors (*n*) is a one-word binary integer. Only the first *n* lock descriptors are processed If *n* is zero, DBLOCK returns without taking any action. The format of a lock descriptor is illustrated in figure 4-5, and the lock descriptor fields are described in table 4-11.1.

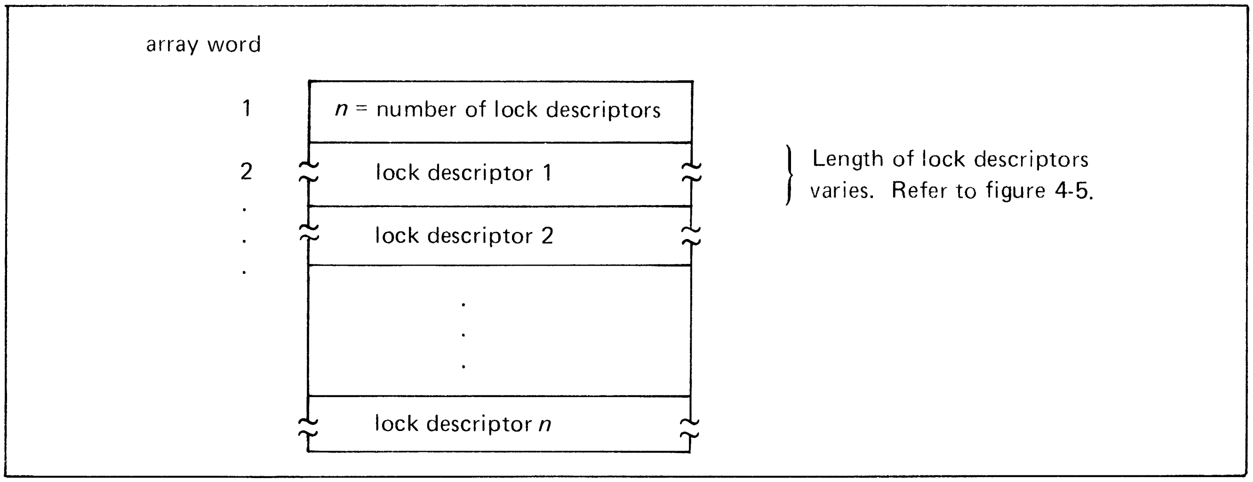
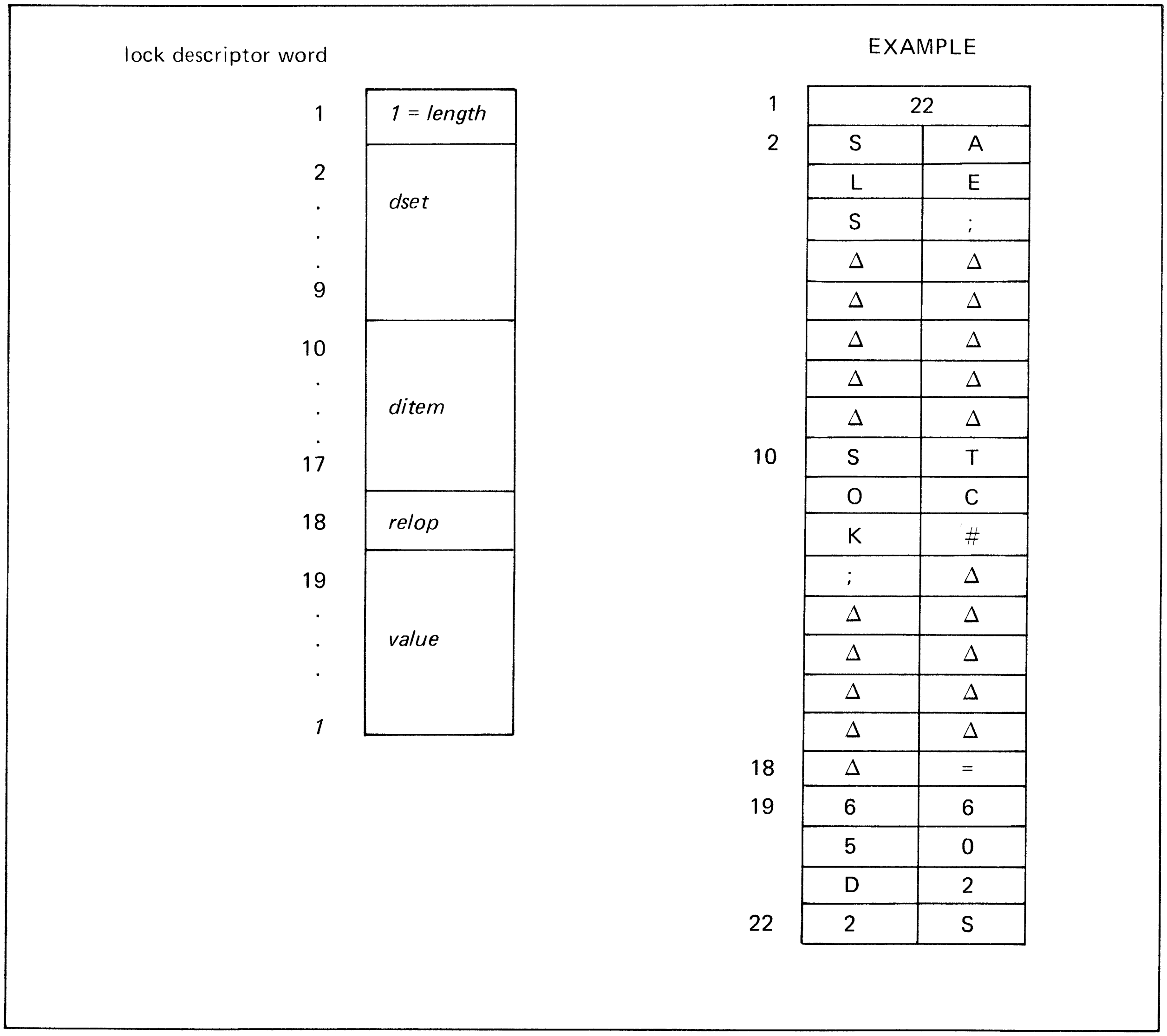

array word
1
2
.
.
.
n = number of lock descriptors
lock descriptor 1
lock descriptor 2
.
.
.
lock descriptor *n*
Length of lock descriptors varies. Refer to figure 4-5.

Figure 4-4. *qualifier* Array Format



1	22	
2	S	A
	L	E
	S	;
	Δ	Δ
	Δ	Δ
	Δ	Δ
	Δ	Δ
	Δ	Δ
10	S	T
	O	C
	K	#
	;	Δ
	Δ	Δ
	Δ	Δ
	Δ	Δ
	Δ	Δ
18	Δ	=
19	6	6
	5	0
	D	2
22	2	S

Figure 4-5. Lock Descriptor Format

Table 4-11.1. Lock Descriptor Fields

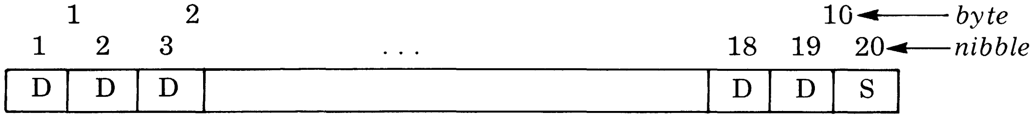
FIELD NAME	DESCRIPTION
length	is a one-word binary integer specifying the physical length in words of the lock descriptor, including the *length* field itself.
dset	is always 8 words long and describes the data set in which locks are placed. It may be one of the following: ● A data set name, left-justified, 16 characters long or, if shorter, terminated with a blank or semicolon. For example: SALES; ● A data set number, a binary integer in the range of 1 to 99 stored in the first word. ● An at-sign (@) stored in the first byte indicating that the whole data base is to be locked. All unused bytes are ignored. In this case, the *ditem, relop,* and *value* fields are ignored and may be omitted if desired. ● A blank or semicolon (first byte) or binary zero (first word) indicating that the whole lock descriptor is to be ignored. (It is counted as one of the *n* descriptors.) The data set, if specified, need not be accessible for read or write access to the user requesting the lock.
ditem	is always 8 words long. It may be one of the following: ● A data item name, left-justified, 16 characters long or, if shorter, terminated with a blank or semicolon. ● A data item number stored as a binary integer in the first word. It may be in the range of 1 to 255. ● An at-sign (@) stored in the first byte indicating that the whole data set specified in *dset* is to be locked. All unused bytes are ignored and may be omitted if desired. The data item need not be a search item, nor does it have to be accessible to the user requesting the lock. However, it cannot be a compound item or a P-type item longer than P28.
relop	is one word long and contains one of the three relational operators represented as two ASCII characters: ● <= less than or equal ● >= greater than or equal ● =Δ or Δ= equal (Δ indicates space character)
value	is the value of the data item to be locked. It must be stored in exactly the same way as it is stored in the data base. IMAGE extracts as many words as required by the corresponding data item definition (in the schema). The rest (if any) are ignored.

Note that the shortest possible descriptor is 9 words long consisting of the *length* field and a *dset* field containing @. Although the *dset* field only contains an at-sign, it must still be 8 words long. The length of the entire descriptor array may not exceed 2047 words.

VALIDITY CHECKING OF DATA ITEM VALUES

If you specify a data item of type P, U, or Z in a lock descriptor, IMAGE checks that the value is valid for that data item type. The following checks are made:

- If the data item is type P, the right half of the rightmost byte must contain a sign and all preceding nibbles must contain decimal digits represented in Binary Coded Decimal (BCD) format. For example, if a data item is defined as type P with a length of 20, the format must be:



D = digit S = sign

This would be declared in COBOL as 19 digits plus a sign or 20 nibbles (P20 in the schema):

```
S9(19)     COMP-3
```

Type P data item used in a lock descriptor may not exceed 28 nibbles (7 words) in length. The locking system treats all sign digits other than 1101_2 as identical. 1101_2 is assumed to be a negative sign.

- If the data item is type U, the value must not contain any lowercase alphabetic characters in the range of *a* through *z*.

- If the data item is type Z, each byte preceding the last one must contain an 8-bit digit represented in ASCII format and the last byte must contain a value representing a digit and a sign. (Refer to the description of packed decimal numbers in Section III of the *Machine Instruction Set Manual.*)

Table 4-12. DBLOCK Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
–7	FLOCK failure.
CALLING ERRORS:	
–11	Bad *base* parameter.
–31	Bad *mode* value
–120	Not enough stack to perform DBLOCK.
–121	Descriptor count error.
–122	Descriptor list bad. Is not entirely within stack.
–123	Illegal *relop* in a descriptor.

Table 4-12. DBLOCK Condition Word Values (Continued)

–124	Descriptor too short. Must be greater than or equal to 9.	
–125	Bad set name/number.	
–126	Bad item name/number.	
–127	Attempt to lock using a compound item.	
–128	Value field too short in a descriptor.	
–129	P-type item longer than P28 specified.	
–130	Illegal digit in a P-type value.	
–131	Lowercase character in type U value.	
–132	Illegal digit in type Z value.	
–133	Illegal sign in type Z value.	
–134	Two descriptors conflict.	
–135	Second lock without CAP=MR.	
–136	Descriptor list exceeds 2047 words.	
COMMUNICATIONS ERRORS:		
–102	DSWRITE failure.	
–103	Remote stack too small.	
–106	Remote data inconsistent.	
–107	DS procedure call error.	
EXCEPTIONAL CONDITIONS		Applicable Modes
20	Data base locked or contains locks (Status word 3: 0 = data base locked 1 = data set or entries locked)	(2)
22	Data set locked by another process	(4, 6)
23	Entries locked within set	(4)
24	Item conflicts with current locks	(6)
25	Entry or entries already locked	(6)
62	DBCB full*	(3, 4, 5, 6)
63	Bad DBCB.	

Appendix A contains more information about these conditions.

TEXT DISCUSSION:
Page 4-13

*Note: If error 62 occurs when multiple lock descriptors are specified, some of the descriptors may have been successfully applied. If so, they are not unlocked by IMAGE before returning the error. It is recommended that a DBUNLOCK be issued after any positive-numbered error unless you have reason to do otherwise.

DBOPEN

INTRINSIC NUMBER 401

Initiates access to the data base and establishes the user class number and access mode for all subsequent data base access.

PARAMETERS

base is the name of a word array containing a string of ASCII characters. The string must consist of a pair of blanks followed by a left-justified data base name and terminated by a semicolon or blank (Δ), for example, ΔΔSTORE;. If the data base is successfully opened, IMAGE replaces the pair of blanks with a value called the *base id.* The *base id* uniquely identifies this access path between the data base and the process calling DBOPEN. In all subsequent accesses to the data base the first word of *base* must be this *base id*; therefore, the array should not be modified.

To access a data base catalogued in a group other than the user's log-on group, the data base name must be followed by a period and the group name; for example, STORE.GROUPX. If the data base is in an account other than the user's account, the group name must be followed by a period and the account name; for example, STORE.GROUPX.ACCOUNT1.

You may use a :FILE command before executing the application program to equate the data base name or the data-base-access file name to another data base or data-base-access file name. Only the format file designator, actual file designator, and the DEV= parameter may be used.

password is the name of a word array containing a left-justified string of ASCII characters, either eight characters long or, if shorter, terminated by a semicolon or blank, for example, DO-ALLΔ. The password establishes a user class number as described in Section II.

mode is an integer between 1 and 8, inclusive, corresponding to the valid IMAGE access modes described earlier in this section. Here is a brief summary:

Access Mode	Associated Capabilities	Concurrent Modes Allowed
1	modify with enforced locking. Allow concurrent modify	1,5
2	update, allow concurrent update	2,6
3	modify exclusive	none
4	modify, allow concurrent read	6
5	read, allow concurrent modify	1,5
6	read, allow concurrent modify	6 and either 2, one 4, or 8.
7	read, exclusive	none
8	read, allow concurrent read	6,8

The table in Appendix B summarizes the results of multiple access to the same data base. If a data base cannot be opened successfully in a particular mode, this table can be used to determine the problem and to select an alternate mode.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. The status array contents are:

Word	Contents
1	Condition word (see table 4-13)
2	User class number, 0 to 63 (or a 64 if data base designer with ";" password)
3	Current word size of the DBCB
4	Word size of the ULCB
5-10	Information about the current procedure call and its results. This same information is returned for all IMAGE procedures if an error occurs. It is described in Appendix A with the summary of condition words.

OPENING A DATA BASE MORE THAN ONCE

A process may concurrently use the data base through independent, unique *access paths* by issuing as many as 63 calls to DBOPEN and specifying different *base* arrays in each call. Subsequent calls to other IMAGE procedures must use the appropriate *base* array so that the correct *base id* is used.

The data base activity controlled on one access path relates to that controlled on other access paths in the same way the data base activity of one process relates to that of another. The access modes established by each DBOPEN call must be compatible but otherwise the activity controlled by each access path and the pointers maintained by it are completely independent. The only exception to this access path independence relates to locking. If a process makes a lock request on one access path it cannot issue a lock on another access path unless the program has multiple RIN capability (CAP=MR) or first calls DBUNLOCK to release the locks on the first access path.

Table 4-13. DBOPEN Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
-1	FOPEN intrinsic failure.
-2	FCLOSE failure.
-3	FREADDIR failure.
-4	FREADLABEL failure.
-9	GETDSEG failure.
CALLING ERRORS:	
-11	Bad *base* parameter.
-21	Bad password.
-31	Bad *mode.*
-32	Unobtainable *mode.*
-91	Bad root modification level.
-92	Data base not created.
-94	Data base bad
COMMUNICATIONS ERRORS:	
-60	Illegal file equation on root file.
-100	DSOPEN failure.
-101	DSCLOSE failure.
-102	DSWRITE failure.
-103	Remote stack too small.
-104	Remote system does not support IMAGE.
-105	MPE intrinsic GETDSEG failure on remote HP 3000.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
61	This data base opened more than 63 times by the same process.
62	DBCB full.
63	Bad DBCB.
64	PCBX data segment area full.
66	The current DBCB for the data base does not appear correct (IMAGE internal error).

Consult Appendix A for more information about these conditions and Appendix B for results of multiple access.

TEXT DISCUSSION:
Page 4-2

DBPUT

INTRINSIC NUMBER 407

Adds new entries to a manual master or detail data set. The data base must be open in access mode 1, 3, or 4.

PARAMETERS

base is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

dset is the name of an array containing the left-justified name of the data set to which the entry is to be added or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or a blank (Δ), for example, CUSTOMER; or SALESΔ.

mode must be an integer equal to 1.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. If the procedure executes successfully, the status array contents are:

Word	Contents
1	Condition word (see table 4-15)
2	Word length of logical entry in *buffer* array
3-4	Doubleword record number of new entry
5-6	Doubleword count of number of entries in chain. If master data set, chain is synonym chain. If detail data set, chain is current chain of new entry.
7-8	If master, doubleword record address of predecessor on synonym chain. If detail, doubleword record number of predecessor on current detail chain.
9-10	If detail, doubleword record number of successor on current chain. If master, doubleword zero.

list is the name of an array containing an ordered set of data item identifiers, either names or numbers. The new entry contains values supplied in the *buffer* array for the data items in the *list* array. Any search or sort items defined for the entry must be included in the *list* array. Fields of unreferenced items are filled with binary zeroes.

The *list* array may contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No embedded blanks are allowed and no name may appear more than once. Example: ACCOUNT,LAST-NAME,CITY,STATE;.

When referencing by number, the first word of the *list* array is an integer *n* which is followed by *n* single integers identifying unique data item numbers. Example: 4 1 10 3 16 lists the four data item numbers 1, 10, 3, and 16.

The *list* specifies the data items for which values are supplied in the *buffer* array and is saved internally by IMAGE as the *current list* for the data set. The *current list* is unchanged until a different list is specified in a subsequent call to DBGET, DBPUT, or DBUPDATE for the same access path and data set.

Some special list constructs are allowed. These are described in table 4-14 and illustrated in the SPL programs in Section V. List processing is a relatively high overhead operation which may be shortened in subsequent calls by using the asterisk construct to specify that the *current list* is to be used.

buffer is the name of an array containing the data item values to be added. The values are concatenated in the same order as their data item identifiers in the *list* array. The number of words for each value must correspond to the number required by its type, for example, 12 values must be 2 words long.

Table 4-14. Special *list* Parameter Constructs

CONSTRUCT	*list* ARRAY CONTENTS	PURPOSE
Empty	0; or 0Δ or ; or Δ (Note: Zero must be ASCII)	Request no data transfer.
Empty Numeric	0 (*n*, length of data item identifier list, is zero)	Request no data transfer.
Asterisk	*; or *Δ	Requests procedure to use previous *list* and apply it to same data set. This construct saves IMAGE processing time, especially if more than one or two items are being dealt with.
Commercial At-Sign	@; or @Δ	Requests procedure to use all data items of the data set in the order of their occurrence in the entry.
Δ indicates blank.		

MASTER DATA SETS

When adding entries to master data sets the following rules apply:

- The data set must be a manual master.
- The search item must be referenced in the *list* array and its value in the *buffer* array must be unique in relation to other entries in the master.
- There must be space in the master set to add an entry.
- The order of data item values in the new entry is determined by the set definition in the schema and not by the order of the items' occurrence in the *list* and *buffer* arrays.
- Unreferenced data items are filled with binary zeroes.
- The caller must have a lock on the data set or the data base if the data base is opened in access mode 1.

Table 4-15. DBPUT Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
-1	FOPEN intrinsic failure.
-3	FREADDIR failure.
-4	FREADLABEL failure.
CALLING ERRORS:	
-11	Bad *base* parameter.
-12	No lock covering entry to be added. (DBOPEN mode 1 only.)
-14	Illegal intrinsic in current access mode.
-21	Bad data set reference.
-23	Data set not writable.
-24	Data set is an automatic master.
-31	Bad *mode.*
-51	Bad *list* length.
-52	Bad *list* or bad *item.*
-53	Missing search or sort item.
COMMUNICATIONS ERRORS:	
-102	DSWRITE failure.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
16	Data set full.
43	Duplicate search item.
62	DBCB full.
63	Bad DBCB.
1*xx*	Missing chain head for path number *xx.*
2*xx*	Full chain for path number *xx.*
3*xx*	Full master for path number *xx.*

Refer to Appendix A for more information about these conditions.

DETAIL DATA SETS

When adding entries to detail data sets the following rules apply:

- The data set must have free space for the entry.
- If the data base is opened in access mode 1, the caller must have a lock covering the entry to be added.
- All search and sort items defined for the entry must be referenced in the *list* array.
- Each related manual master data set must contain a matching entry for the corresponding search item value. If any automatic master does not have a matching entry, it must have space to add one. This addition occurs automatically.*
- The order of data item values in the new entry is determined by the set definition in the schema and not by the order of the items' occurrence in the *list* and *buffer* arrays.
- Unreferenced data items are filled with binary zeroes.
- The new entry is linked into one chain for each search item, or path, defined according to the search item value. It is linked to the end of chains having no sort items and into its sorted position according to the collating sequence of the sort item values in the chain. If two or more entries have the same sort item value, their position in the chain is determined by the values of the items following the sort item in the entry.

  The position of an entry on a sorted chain is determined by a backward search of the chain beginning at the last entry. The position is maintained by logical pointers rather than physical placement in the file.

DATA SET INTERNAL INFORMATION

The record in which the new data entry is placed becomes the current record for the data set. The forward and backward pointers reflect the new entry's position. Refer to the description of *status* words 7 through 10.

*Note: The internally maintained synonym chain pointers for the automatic master are set to zero. (See Section VII).

TEXT DISCUSSION:
Page 4-6

DBUNLOCK

INTRINSIC NUMBER 410

Relinquishes the locks acquired by a previous call to DBLOCK. Redundant calls are ignored. If the calling process has the same data base open multiple times, only those locks put into effect for the specified access path are unlocked.

PARAMETERS

base is the name of the array used for the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

dset is currently unused. Use the DUMMY variable as recommended at the beginning of this section or any *dset* array used for other procedures.

mode must be an integer equal to 1.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. The status array contents are:

Word	Contents
1	Condition word (see table 4-16)
2	Number of lock descriptors released by this call. Each data set lock or data base lock is counted as one descriptor.
3-4	Unchanged from previous call using this array.
5-10	Information about the procedure call and its results. Refer to Appendix A for a description of this information.

Table 4-16. DBUNLOCK Condition Word Values

CALLING ERRORS:	
−11	Bad *base* parameter.
−31	Bad *mode.*
COMMUNICATIONS ERRORS:	
−102	DSWRITE failure.
−106	Remote data inconsistent.
−107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
63	Bad DBCB.

Appendix A contains more information about these conditions.

TEXT DISCUSSION:

Page 4-13

DBUPDATE

INTRINSIC NUMBER 406

Modifies values of data items in the entry residing at the current record address of a specified data set. Search and sort item values cannot be modified. The data base must be open in access mode 1, 2, 3, or 4. The update will always be carried out correctly against the latest version of the data, regardless of modifications that may be made by other users.

PARAMETERS

base is the name of the array used as the *base* parameter when opening the data base. The first word of the array must contain the *base id* returned by DBOPEN.

dset is the name of an array containing the left-justified name of the data set to be read or is an integer referencing the data set by number. The data set name may be 16 characters long or, if shorter, terminated by a semicolon or blank.

mode must be an integer equal to 1.

status is the name of a ten-word array in which IMAGE returns status information about the procedure. If the procedure operates successfully, the status array contents are:

Word	Contents
1	Condition word (see table 4-17)
2	Word length of the values in buffer
3-10	Same doubleword values set by preceding procedure call which positioned the data set at the current entry.

list is the name of an array containing an ordered set of data item identifiers, either names or numbers. Values supplied in the *buffer* array replace the values of data items occupying the same relative position in the *list* array. The user class established when the data base is opened must allow at least read access to all the items included in the *list* array. If the corresponding *buffer* array values are the same as the current data item values, the *list* array can include data items the user can read but is not permitted to alter. This feature permits reading and updating with the same *list* array contents.

The *list* array may contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No embedded blanks are allowed and no name may appear more than once.

When referencing by number, the first word of the *list* array is an integer *n* followed by *n* unique data item numbers (one-word integers).

The *list* not only specifies the data items to be updated immediately but is saved internally by IMAGE as the *current list* for this data set. The *current list* is unchanged until a different list is specified in a subsequent call to DBGET, DBPUT, or DBUPDATE for the same access path and data set.

Some special list constructs are allowed. These are described in table 4-14 with the DBPUT procedure. List processing is a relatively high overhead operation which may be shortened substantially in subsequent calls by using the asterisk construct to specify that the *current list* is to be used.

Table 4-17. DBUPDATE Condition Word Values

FILE SYSTEM AND MEMORY MANAGEMENT FAILURES:	
-1	FOPEN intrinsic failure.
-3	FREADDIR failure.
-4	FREADLABEL failure.
CALLING ERRORS:	
-11	Bad *base* parameter.
-12	No locks cover entry to be updated. (DBOPEN mode 1 only.)
-14	Illegal intrinsic in current access mode.
-21	Bad data set reference.
-31	Bad *mode*
-51	Bad *list* length.
-52	Bad *list* or bad *item.*
COMMUNICATIONS ERRORS:	
-102	DSWRITE failure.
-106	Remote data inconsistent.
-107	DS procedure call error.
EXCEPTIONAL CONDITIONS:	
17	No entry.
41	Critical item.
42	Read only item.
62	DBCB full.
63	Bad DBCB.

Appendix A contains more information about these conditions.

buffer is the name of an array containing concatenated values to replace the values of data items occupying the same relative position in the *list* array. The number of words for each value must correspond to the number of words required by its type multiplied by the sub-item count.

LOCKING

Before performing an update for a data base opened in access mode 1, IMAGE verifies that locks are in effect to cover the data entry both before and after it is modified.

DATA SET INTERNAL INFORMATION

The current record number, forward and backward pointers are unchanged. (Refer to the description of *status* words 3 through 10.)

TEXT DISCUSSION:

Page 4-12

LANGUAGE CONSIDERATIONS AND EXAMPLES

SECTION V

This section is divided into five separate discussions, each covering the use of IMAGE with a specific programming language: COBOL, FORTRAN, SPL, BASIC, and RPG.

The examples in each language are designed to illustrate simply and directly the way IMAGE procedures are cailed. They are not intended as models of the best way to code the task which is illustrated since this will vary with the application requirements and an individual programmer's coding methods.

A knowledge of the programming language is assumed. If you have questions about the language itself, consult the appropriate language manual:

COBOL/3000 Reference Manual
FORTRAN Reference Manual
System Programming Language Reference Manual
BASIC Interpreter Reference Manual
BASIC/3000 Compiler Reference Manual
RPG/3000 Compiler Reference and Application Manual

All examples presented in this section perform operations on the STORE data base. Figures 2-5 and 2-6 in Section II and figure 3-5 in Section III should be consulted if questions about the data base structure arise in relation to the examples.

COBOL EXAMPLES

To illustrate the use of IMAGE procedures through COBOL programs, sample lines of code that perform a specific task are given. The IMAGE procedure calling parameters are described by the way they are defined in the data division and their value when the procedure is called or, in some cases, after it is executed.

The BASE-NAME record is described only in the first two examples. Once the data base has been opened and the data segment number has been moved to the first word as shown in the ADD ENTRY example, it remains the same for all subsequent calls illustrated.

The STATUSS record is defined in the same way for all examples but its content varies depending upon which procedure is called and the results of that procedure. The STATUSS record is defined as:

```
01  STATUSS.
    05  CONDTN-WORD                 PIC 9999  COMP.
    05  STAT1                       PIC 9999  COMP.
    05  STAT2-3                     PIC 9(9)  COMP.
    05  STAT4-5                     PIC 9(9)  COMP.
    05  STAT6-7                     PIC 9(9)  COMP.
    05  STAT8-9                     PIC 9(9)  COMP.
```

The DUMMY parameter appears as a reminder when a parameter is not used by a procedure performing the task being illustrated. DUMMY can be defined as PIC 9999 COMP.

When the code GOTO ASK-FOR-IP appears, it indicates that the program continues and prompts the user for further instructions, for example, it may request the type of data base operation the user wants to perform.

OPEN DATA BASE

```
PROCEDURE DIVISION.
FIRST-PARAGRAPH-NAME.
    CALL "DBOPEN" USING BASE-NAME, PASSWORD, MODE2, STATUSS.
    IF CONDTN-WORD NOT = 0 DISPLAY "DBOPEN-FAIL "
      CALL "DBEXPLAIN" USING STATUSS  STOP  RUN.
```

Parameter	Definition	Value
BASE-NAME	PIC X(8)	"ΔΔSTORE;"
PASSWORD	PIC X(8)	"DO-ALL;Δ" or "DO-ALLΔΔ"
MODE3	PIC 9999 COMP	3

In this example, the STORE data base is opened in access mode 3 with the password DO-ALL that establishes user class number 18. The value of PASSWORD may be specified in the data division or it may be requested from the application program user and moved into PASSWORD. If the password is fewer than 8 characters it must be followed by a blank or semi-colon. In this program, the first word of the STATUSS array, CONDTN-WORD, is tested and if it is not zero a failure message is printed and the DBEXPLAIN procedure is executed.

ADD ENTRY

```
CALL "DBPUT" USING BASE-NAME, DATA-SET-P, MODE1,
  STATUSS, ALL-ITEMS, PR-BUFFER.
IF CONDTN-WORD = 43 DISPLAY "DUPLICATE STOCK NUMBER"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD = 16 DISPLAY "DATA SET FULL"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD = -23 DISPLAY "CANNOT ADD WITH CURRENT PASSWORD"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

Parameter	Definition	Value
BASE-NAME	PIC X(8)	“ 23 STORE;” (data segment number in first word)
DATA-SET-P	PIC X(8)	“PRODUCT;”
MODE1	PIC 9999 COMP	1
ALL-ITEMS	PIC X(2)	“@;”
PR-BUFFER		
STOCK-NO DESCRIPTN	PIC X(8) PIC X(20)	“7474Z74Z” “ORANGE CRATEΔΔΔΔΔΔΔΔ”

This sample code adds a data entry to the PRODUCT manual master data set. Note that the first word of BASE-NAME now contains the number of the privileged data segment of the Data Base Control Block. ALL-ITEMS contains an at-sign indicating that PR-BUFFER contains a value for all items in the data entry. The values for the STOCK# and DESCRIPTION data items are concatenated in PR-BUFFER.

A program may be designed to prompt for both the data set name and the data item values that are moved into PR-BUFFER and added to the data set. In the example, the condition word of the status array is tested for a value of 43, indicating that an entry with the search item value 7474Z74Z already exists in the data set, or 16, indicating that the data set is full. If the user class is not in the data set write class list, a condition word of –23 is returned.

If an entry is to be added to a detail set, the program may first check to see if the required entries exist in the manual masters linked to the detail set. Values must be provided for all search items and the sort item, if one is defined, of a detail data set entry.

READ ENTRY (SERIALLY)

```
READ-NEXT.
    CALL "DBGET" USING BASE-NAME, DATA-SET-C, MODE2, STATUSS,
      LIST-OF-ITEMS, CU-BUFFER, DUMMY.
    IF CONDTN-WORD = 11 PERFORM REWIND
      GO TO READ-NEXT.
    IF CONDTN-WORD = -21 DISPLAY "NO READ ACCESS TO DATA"
      GO TO ASK-FOR-IP.
    IF CONDTN-WORD NOT = 0 GO TO DISPLAY-STATUS.
                              .
        (process entry and decide whether or not to continue)
```

Parameter	Definition	Value
DATA-SET-C	PIC X(10)	"CUSTOMER; "
MODE2	PIC 9999 COMP	2
LIST-OF-ITEMS	PIC X(80)	"ACCOUNT,FIRST-NAME,LAST-NAME;.."
CU-BUFFER		
ACCT	PIC 9(9) COMP	12345678
F-NAME	PIC X(10)	"GEORGE" *Values which are read.*
L-NAME	PIC X(16)	"PADERSON"

To read the next entry of the CUSTOMER data set, a mode of 2 is used. This directs the DBGET procedure to perform a forward serial read. In the example, LIST-OF-ITEMS contains the names of three data items. After DBGET returns to the calling program, CU-BUFFER contains the values shown. If an end-of-file is encountered the condition word is set to 11. In this case, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the DBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read will read the first entry in the data set.

If the condition word –21 is returned, the user's password does not grant read access to data.

The DUMMY variable merely signifies that the *argument* parameter is not used with mode 2.

READ ENTRY (DIRECTLY)

```
CALL "DBGET" USING BASE-NAME, DATA-SET-I, MODE4, STATUSS,
  ALL-ITEMS, IN-BUFFER, RECORD-NUMBER.
IF CONDTN-WORD = 12 OR = 13 DISPLAY "INCORRECT RECORD NUMBER"
  GO TO DISPLAY-STATUS.
IF CONDTN-WORD = 17 DISPLAY "RECORD CONTAINS NO DATA ENTRY"
  GO TO DISPLAY-STATUS.
IF CONDTN-WORD NOT = 0 DISPLAY "DBGET FAILURE"
  GO TO DISPLAY-STATUS.
                        .
                        .
```

Parameter	Definition	Value
DATA-SET-I	PIC X(10)	“INVENTORY;”
MODE4	PIC 9999 COMP	4
ALL-ITEMS	PIC X(2)	“@;”
RECORD-NUMBER	PIC 9(9) COMP	33
IN-BUFFER		
STOCK-NO-I	PIC X(8)	“3333A33A”
QTY	PIC 9(9) COMP	452
SUPPLIER	PIC X(16)	“H & S SURPLUS ”
UNIT-COST	PIC S9(7) COMP-3	0000349E (3495 in 8 nibbles)
LASTSHIPDATE	PIC X(6)	“760824”
BINNUM	PIC X(2)	“03”

The code in this example reads all data items of the entry in record number 33 of the INVENTORY data set using a directed read, mode 4. The program may have saved the record number while reading down the chain of all data entries with STOCK# equal to 3333A33A looking for the latest LASTSHIPDATE. It then reads all data items of the entry which has the desired last shipping date. It is more efficient to read it directly than to search down the chain again.

If the record number is less than 1, the condition word is set to 12. If it is greater than the highest numbered record in the data set, the condition word is set to 13. The condition word is 17 if the record contains no data entry.

READ ENTRY (CALCULATED)

```
CALL "DBGET" USING BASE-NAME, DATA-SET-P, MODE7, STATUSS,
  LIST-OF-ITEMS, DESCRIPTN, STOCK-SEARCH.
IF CONDTN-WORD = 17 DISPLAY "NO SUCH STOCK NUMBER"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD = -21 DISPLAY "NO READ ACCESS TO DATA"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

Parameter	Definition	Value
DATA-SET-P	PIC X(8)	“PRODUCT;”
MODE7	PIC 9999 COMP	7
LIST-OF-ITEMS	PIC X(80)	“DESCRIPTION;”
PR-BUFFER		
STOCK-NO	PIC X(8)	– – –
DESCRIPTN	PIC X(20)	“CLIPBOARD ”
STOCK-SEARCH	PIC X(8)	“2222B22B”

COBOL

To locate the PRODUCT data set entry which has STOCK# search item value of 2222B22B, a calculated read is used. The mode is 7 and the item to be read is DESCRIPTION. After DBGET returns control to the calling program, the description of stock number 2222B22B is in DESCRIPTN. If no entry exists with STOCK# equal to 2222B22B, the condition word is 17. If the user does not have read access to the DESCRIPTION data item, condition word −21 is returned.

READ ENTRY (BACKWARD CHAIN)

```
    CALL "DBFIND" USING BASE-NAME, DATA-SET-S, MODE1, STATUSS,
      ITEM-NAME, ITEM-VALUE.
    IF CONDTN-WORD = 17 DISPLAY "NO PURCHASES ON THAT DATE"
      GO TO ASK-FOR-IP.
    IF CONDTN-WORD = -21 OR -52
      DISPLAY "PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS"
      GO TO ASK-FOR-IP.
    IF CONDTN-WORD NOT = 0 DISPLAY "DBFIND FAILURE"
      GO TO DISPLAY-STATUS.
NEXT-IN-CHAIN.
    CALL "DBGET" USING BASE-NAME, DATA-SET-S, MODE6, STATUSS,
      ALL-ITEMS, SA-BUFFER, DUMMY.
    IF CONDTN-WORD = 14 DISPLAY "NO MORE PURCHASES ON THIS DATE"
      GO TO NEXT-ACCOUNT
    IF CONDTN-WORD = 0  GO TO REPORT-SALES.
                  .
                  .
REPORT-SALES.

        (routine to print sales information)

    GO TO NEXT-IN-CHAIN.
```

Parameter	Definition	Value
DATA-SET-S	PIC X(6)	"SALES;"
MODE1	PIC 9999 COMP	1
ITEM-NAME	PIC X(12)	"PURCH-DATE; "
ITEM-VALUE	PIC X(6)	"760314"
MODE6	PIC 9999 COMP	6
ALL-ITEMS	PIC X(2)	"@;"
SA-BUFFER		
ACCOUNT-S	PIC (9) COMP	12345678
STOCK-NO-S	PIC X(8)	"2222B22B"
QUANTITY	PIC 9999 COMP	3
PRICE	PIC 9(9) COMP	425 *Sample values read from one entry in chain.*
TAX	PIC 9(9) COMP	25
TOTAL	PIC 9(9) COMP	450
PURCH-DATE	PIC X(6)	"760314"
DELIV-DATE	PIC X(6)	"760320"

First the DBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value 760314 of both the master entry containing the chain head and the detail entries making up the chain. If no entry in the DATE-MASTER has a search item value of 760314, the condition word will be 17. If the user's password or access mode does not allow read access to the data, condition word –21 or –52 is returned.

If the DBFIND procedure executes successfully, a call to the DBGET procedure with a mode parameter of 6 reads the last entry in the chain. Successive calls to DBGET with the same mode read the next-to-last entry and so forth until the first entry in the chain has been read. A subsequent call to DBGET returns condition word 14, indicating the beginning of the chain has been reached and no more entries are available. If an entry has been successfully read, the program executes the REPORT-SALES routine and prints the information. It then goes to the NEXT-IN-CHAIN routine and reads another entry.

If no entries exist in the chain, the condition word is also 14.

UPDATE ENTRY

```
CALL "DBGET" USING BASE-NAME, DATA-SET-C, MODE7, STATUSS,
   ITEM-NAME, ADDRESS-VALUE, ACCT-SEARCH.

    (Determine if entry successfully read, print current
     address, and prompt for new address.)

CALL "DBUPDATE" USING BASE-NAME, DATA-SET-C, MODE1, STATUSS,
   ITEM-NAME, ADDRESS-VALUE.
IF CONDTN-WORD = 42 DISPLAY "NOT ALLOWED TO ALTER THIS ITEM"
   GO TO ASK-FOR-IP.
IF CONDTN-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

Parameter	Definition	Value
DATA-SET-C	PIC X(10)	“CUSTOMER; ”
MODE7	PIC 9999 COMP	7
MODE1	PIC 9999 COMP	1
ACCT-SEARCH	PIC 9(9) COMP	12345678
ITEM-NAME	PIC X(16)	“STREET-ADDRESS; ”
ADDRESS-VALUE	PIC X(26)	“12 SUTTON PLACE ”

In order to update an entry it must first be located. In this example, the entry is located by using a calculated DBGET to read the STREET-ADDRESS item in the CUSTOMER data set. The entry is located by using the ACCOUNT search item with a value of 12345678. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into ADDRESS-VALUE. The DBUPDATE routine is then called to alter the STREET-ADDRESS data item in the entry.

If the current user class number does not allow this item to be altered or the access mode does not allow updates to take place, the condition word 42 is returned.

A null list can be used when calling DBGET to locate an entry to be updated.

DELETE ENTRY

```
(Locate appropriate entry as with DBGET.)

CALL "DBDELETE" USING BASE-NAME, DATA-SET-C, MODE1, STATUSS.
IF CONDTN-WORD = 44
  DISPLAY "SALES ENTRIES EXIST, CANNOT DELETE CUSTOMER"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD = -23 DISPLAY "PASSWORD DOES NOT ALLOW DELETE"
  GO TO ASK-FOR-IP.
IF CONDTN-WORD NOT = 0 GO TO DISPLAY-STATUS.
```

Parameter	Definition	Value
DATA-SET-C	PIC X(10)	"CUSTOMER; "

Before an entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling DBGET. In this example, the program may have requested the account number of the customer to be deleted and then used a calculated DBGET to locate the appropriate entry. If entries in the SALES data set exist which have the same account number as the entry to be deleted, the condition word is set to 44 and the entry is not deleted. Condition word –23 indicates that the user does not have the capability of deleting an entry from the CUSTOMER data set.

A null list can be used when calling DBGET to locate an entry to be deleted.

LOCK AND UNLOCK (DATA BASE)

```
    CALL "DBLOCK" USING BASE-NAME, DUMMY, MODE2, STATUSS.
    IF CONDTN-WORD = 20 DISPLAY "DATA BASE IS BUSY. TRY AGAIN LATER."
      GO TO CLOSE.
    IF CONDTN-WORD = 0 GO TO USE-BASE.
    DISPLAY "DBLOCK FAILURE" GO TO DISPLAY-STATUS.
USE-BASE.
                        .
                        .
    CALL "DBUNLOCK" USING BASE-NAME, DUMMY, MODE1, STATUSS.
    IF CONDTN-WORD NOT = 0 DISPLAY "DBUNLOCK FAILURE"
      GO TO DISPLAY-STATUS.
```

Parameter	Definition	Value
MODE2	PIC 9999 COMP	2
MODE1	PIC 9999 COMP	1

In this example the program calls DBLOCK to lock the data base. Since mode 2 is used, the program must check the condition word when DBLOCK returns control to verify that the data base is locked. If it is locked the condition word is 0; if it is busy the condition word is 20.

If the data base is successfully locked, the program goes to the USE-BASE routine. After the data base operations have been completed, the program unlocks the data base by calling the DBUNLOCK procedure.

An example of data entry locking appears in the sample COBOL program, figure 5-1.

REQUEST DATA ITEM INFORMATION

```
CALL "DBINFO" USING BASE-NAME, ITEM-NAME, MODE, STATUSS,
  INFO-BUFFER.
IF CONDTN-WORD NOT = 0 DISPLAY "DBINFO FAILURE"
  GO TO DISPLAY-STATUS.
```

Parameter	Definition	Value
ITEM-NAME	PIC X(12)	“PURCH-DATE; ”
MODE	PIC 9999 COMP	102
INFO-BUFFER		
NAM-TYP	PIC X(18)	“PURCH-DATE X ”
SUB-LENG	PIC 9999 COMP	6
SUB-COUNT	PIC 9999 COMP	1

The procedure call in this example obtains information about the PURCH-DATE data item by specifying mode 102. The item name and type are returned in the first 9 words of INFO-BUFFER and the sub-item length and sub-item count in words 10 and 11.

REWIND DATA SET

```
REWIND.
    CALL "DBCLOSE" USING BASE-NAME, DATA-SET-C, MODE3, STATUSS.
    IF CONDTN-WORD NOT = 0 DISPLAY "DBCLOSE FAILURE"
      GO TO DISPLAY-STATUS.
                              •
                              •
```

Parameter	Definition	Value
DATA-SET-C	PIC X(10)	“CUSTOMER; ”
MODE3	PIC 9999 COMP	3

To rewind the CUSTOMER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the Data Set Control Block for CUSTOMER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request will read the first entry in the data set.

CLOSE DATA BASE

```
CLOSE.
    CALL "DBCLOSE" USING BASE-NAME, DUMMY, MODE1, STATUSS.
    IF CONDTN-WORD NOT = 0 CALL "DBEXPLAIN" USING STATUSS.
    STOP  RUN.
```

Parameter	Definition	Value
MODE1	PIC 9999 COMP	1

This call closes the data base. It is issued after the program has completed all data base activity and before program termination.

PRINT ERROR

```
DISPLAY-STATUS.
    CALL "DBEXPLAIN" USING STATUSS.
    GO TO CLOSE.
```

The call to DBEXPLAIN prints a message on the $STDLIST device which interprets the contents of the STATUSS array. This routine may be used while debugging the application if a procedure call fails.

MOVE ERROR TO BUFFER

```
CALL "DBERROR" USING STATUSS, ERR-BUFFER, LENGTH.
```

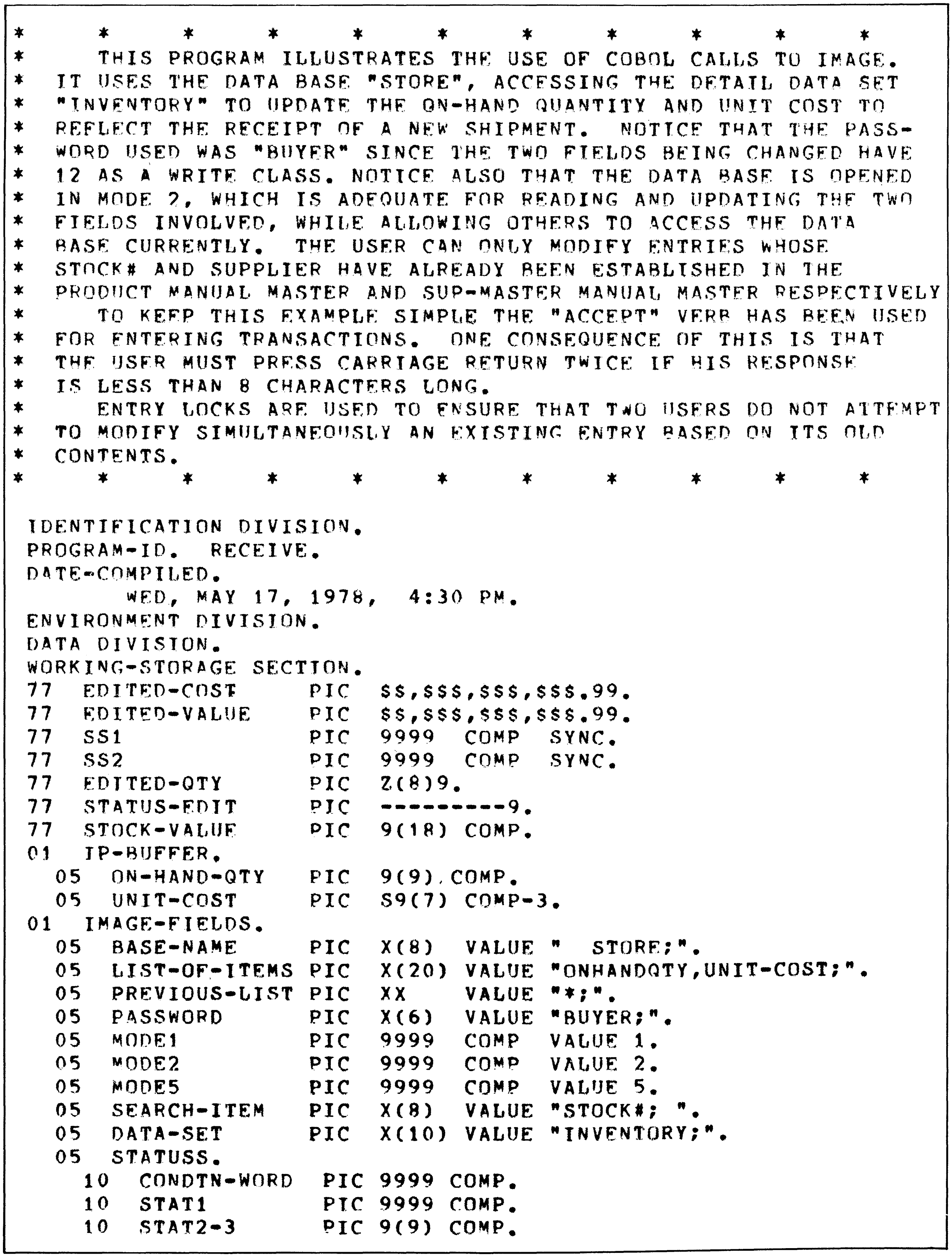
Parameter	Definition	Value
ERR-BUFFER	PIC X(36)	"DATA BASE IN USE "
LENGTH	PIC 9999 COMP	16

In this example, a call to DBERROR has returned one of the messages appropriate when the condition word is equal to –1. The length of the message is 16 bytes as indicated by the value of LENGTH returned by DBERROR.

SAMPLE COBOL PROGRAM

Figure 5-1 contains a sample data base application, a program to update the inventory records, which is coded in COBOL. The program is called RECEIVE and updates on-hand quantities and adjusts unit costs in the INVENTORY data set of the STORE data base. The data base is opened in mode 2. Sample output from RECEIVE is illustrated in figure 5-2.

Locking is performed at the data entry level to ensure that two users do not attempt to modify the same data entry simultaneously.

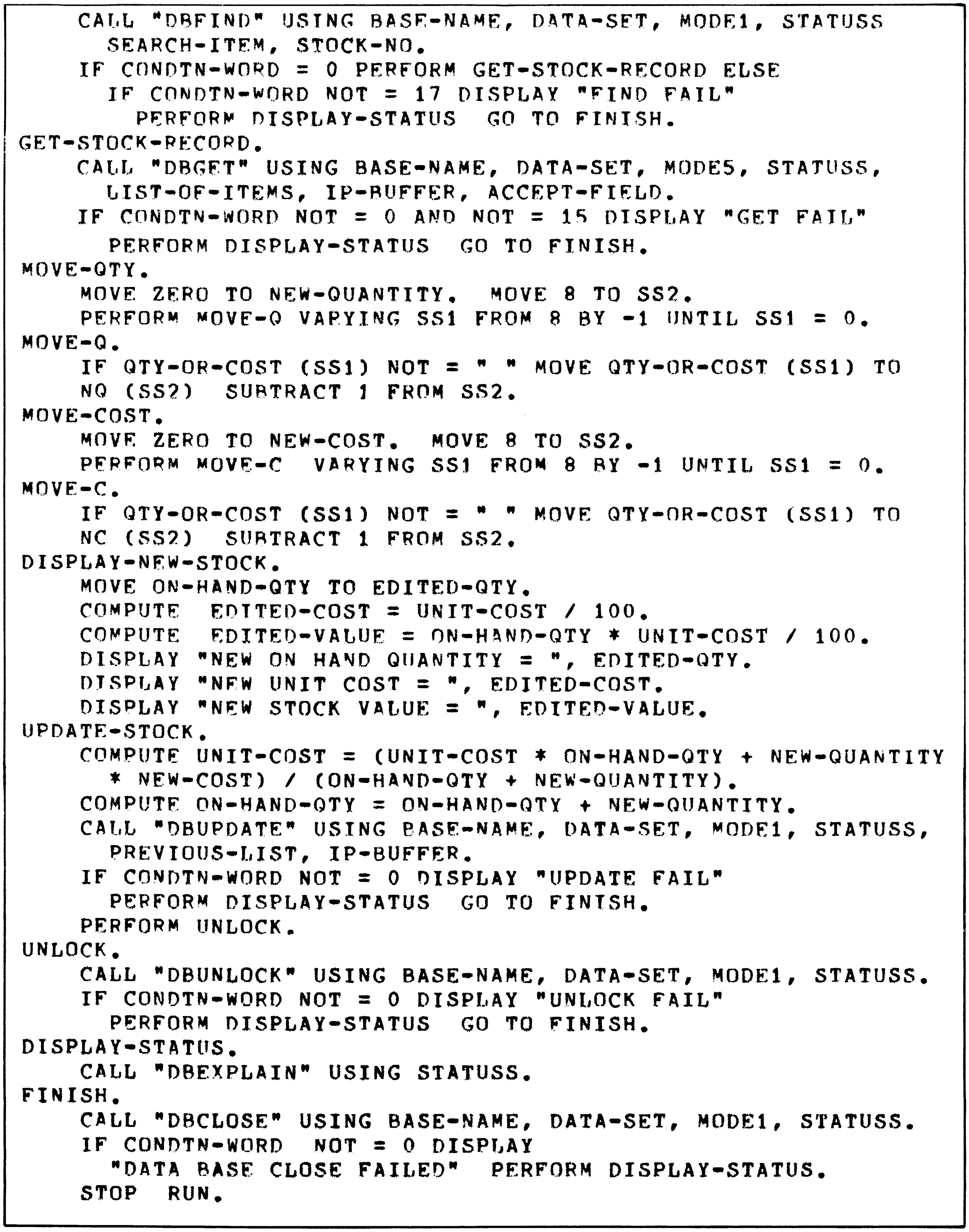


```
*     *     *     *     *     *     *     *     *     *     *
*     THIS PROGRAM ILLUSTRATES THE USE OF COBOL CALLS TO IMAGE.
*  IT USES THE DATA BASE "STORE", ACCESSING THE DETAIL DATA SET
*  "INVENTORY" TO UPDATE THE ON-HAND QUANTITY AND UNIT COST TO
*  REFLECT THE RECEIPT OF A NEW SHIPMENT.  NOTICE THAT THE PASS-
*  WORD USED WAS "BUYER" SINCE THE TWO FIELDS BEING CHANGED HAVE
*  12 AS A WRITE CLASS. NOTICE ALSO THAT THE DATA BASE IS OPENED
*  IN MODE 2, WHICH IS ADEQUATE FOR READING AND UPDATING THE TWO
*  FIELDS INVOLVED, WHILE ALLOWING OTHERS TO ACCESS THE DATA
*  BASE CURRENTLY.  THE USER CAN ONLY MODIFY ENTRIES WHOSE
*  STOCK# AND SUPPLIER HAVE ALREADY BEEN ESTABLISHED IN THE
*  PRODUCT MANUAL MASTER AND SUP-MASTER MANUAL MASTER RESPECTIVELY
*     TO KEEP THIS EXAMPLE SIMPLE THE "ACCEPT" VERB HAS BEEN USED
*  FOR ENTERING TRANSACTIONS.  ONE CONSEQUENCE OF THIS IS THAT
*  THE USER MUST PRESS CARRIAGE RETURN TWICE IF HIS RESPONSE
*  IS LESS THAN 8 CHARACTERS LONG.
*     ENTRY LOCKS ARE USED TO ENSURE THAT TWO USERS DO NOT ATTEMPT
*  TO MODIFY SIMULTANEOUSLY AN EXISTING ENTRY BASED ON ITS OLD
*  CONTENTS.
*     *     *     *     *     *     *     *     *     *     *

 IDENTIFICATION DIVISION.
 PROGRAM-ID.  RECEIVE.
 DATE-COMPILED.
        WED, MAY 17, 1978,  4:30 PM.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  EDITED-COST     PIC  $$,$$$,$$$,$$$.99.
 77  EDITED-VALUE    PIC  $$,$$$,$$$,$$$.99.
 77  SS1             PIC  9999  COMP  SYNC.
 77  SS2             PIC  9999  COMP  SYNC.
 77  EDITED-QTY      PIC  Z(8)9.
 77  STATUS-EDIT     PIC  ---------9.
 77  STOCK-VALUE     PIC  9(18) COMP.
 01  IP-BUFFER.
   05  ON-HAND-QTY   PIC  9(9) COMP.
   05  UNIT-COST     PIC  S9(7) COMP-3.
 01  IMAGE-FIELDS.
   05  BASE-NAME     PIC  X(8)  VALUE "  STORE;".
   05  LIST-OF-ITEMS PIC  X(20) VALUE "ONHANDQTY,UNIT-COST;".
   05  PREVIOUS-LIST PIC  XX    VALUE "*;".
   05  PASSWORD      PIC  X(6)  VALUE "BUYER;".
   05  MODE1         PIC  9999  COMP  VALUE 1.
   05  MODE2         PIC  9999  COMP  VALUE 2.
   05  MODE5         PIC  9999  COMP  VALUE 5.
   05  SEARCH-ITEM   PIC  X(8)  VALUE "STOCK#; ".
   05  DATA-SET      PIC  X(10) VALUE "INVENTORY;".
   05  STATUSS.
     10  CONDTN-WORD  PIC 9999 COMP.
     10  STAT1        PIC 9999 COMP.
     10  STAT2-3      PIC 9(9) COMP.
```

Figure 5-1. Inventory Update Program

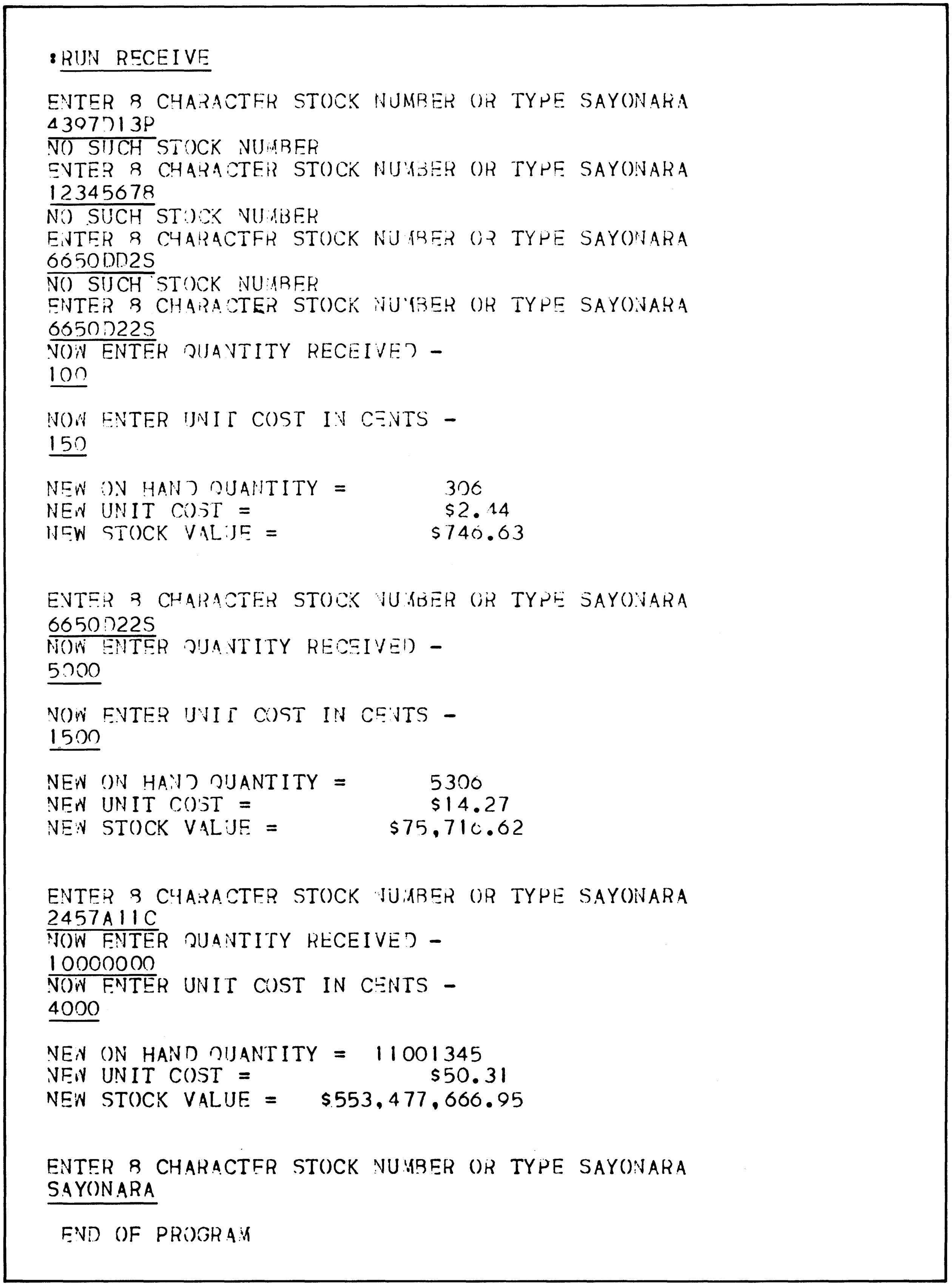
```
    10  STAT4-5        PIC 9(9) COMP.
    10  STAT6-7        PIC 9(9) COMP.
    10  STAT8-9        PIC 9(9) COMP.
01  ACCEPT-FIELD.
  05  STOCK-NO       PIC X(8).
  05  QTY-OR-COST    REDEFINES STOCK-NO  OCCURS 8 PIC X.
  05  FILLER         PIC X    VALUE ";".
01  FILLER.
  05  NEW-QUANTITY  PIC 9(8).
  05  NQ  REDEFINES NEW-QUANTITY  PIC X OCCURS 8.
  05  NEW-COST      PIC 9(8).
  05  NC  REDEFINES NEW-COST      PIC X OCCURS 8.
01  DESCRIPT.
    05  NUM           PIC  S9(4)  COMP  VALUE 1.
    05  LOCK-STOCK-ENTRY.
        10  LENGTHWD  PIC  S9(4)  COMP  VALUE 22.
        10  SETNAME   PIC  X(16)  VALUE "INVENTORY;      ".
        10  ITEMNAME  PIC  X(16)  VALUE "STOCK#;         ".
        10  RELOP     PIC  X(2)   VALUE "= ".
        10  LOCK-VAL  PIC  X(8).

PROCEDURE DIVISION.
FIRST-PARAGRAPH-NAME.
    CALL "DBOPEN" USING BASE-NAME, PASSWORD, MODE2, STATUSS.
    IF CONDTN-WORD NOT = 0 DISPLAY "DBOPEN-FAIL "
      PERFORM DISPLAY-STATUS  STOP  RUN.
ASK-FOR-IP.
   MOVE SPACES TO STOCK-NO.
   DISPLAY "ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA".
   ACCEPT STOCK-NO.
   IF STOCK-NO = "SAYONARA" GO TO FINISH.
   PERFORM FIND-STOCK-RECORD.
   IF CONDTN-WORD = 17 OR = 15
     DISPLAY "NO SUCH STOCK NUMBER"
     PERFORM UNLOCK
     GO TO ASK-FOR-IP.
   MOVE SPACES TO STOCK-NO.
   DISPLAY "NOW ENTER QUANTITY RECEIVED - ".
   ACCEPT STOCK-NO.
   PERFORM MOVE-QTY.
   MOVE SPACES TO STOCK-NO.
   DISPLAY "NOW ENTER UNIT COST IN CENTS - ".
   ACCEPT STOCK-NO.
   PERFORM MOVE-COST.
   PERFORM UPDATE-STOCK.
   PERFORM DISPLAY-NEW-STOCK.
   DISPLAY " " DISPLAY " "
   GO TO ASK-FOR-IP.
FIND-STOCK-RECORD.
    MOVE STOCK-NO TO LOCK-VAL.
    CALL "DBLOCK" USING BASE-NAME, DESCRIPT, MODE5, STATUSS.
    IF CONDTN-WORD NOT = 0  DISPLAY "LOCK FAILED"
      PERFORM DISPLAY-STATUS  GO TO FINISH.
```

Figure 5-1. Inventory Update Program (Continued)



```
    CALL "DBFIND" USING BASE-NAME, DATA-SET, MODE1, STATUSS
      SEARCH-ITEM, STOCK-NO.
    IF CONDTN-WORD = 0 PERFORM GET-STOCK-RECORD ELSE
      IF CONDTN-WORD NOT = 17 DISPLAY "FIND FAIL"
        PERFORM DISPLAY-STATUS  GO TO FINISH.
GET-STOCK-RECORD.
    CALL "DBGET" USING BASE-NAME, DATA-SET, MODE5, STATUSS,
      LIST-OF-ITEMS, IP-BUFFER, ACCEPT-FIELD.
    IF CONDTN-WORD NOT = 0 AND NOT = 15 DISPLAY "GET FAIL"
      PERFORM DISPLAY-STATUS  GO TO FINISH.
MOVE-QTY.
    MOVE ZERO TO NEW-QUANTITY.  MOVE 8 TO SS2.
    PERFORM MOVE-Q VARYING SS1 FROM 8 BY -1 UNTIL SS1 = 0.
MOVE-Q.
    IF QTY-OR-COST (SS1) NOT = " " MOVE QTY-OR-COST (SS1) TO
    NQ (SS2)  SUBTRACT 1 FROM SS2.
MOVE-COST.
    MOVE ZERO TO NEW-COST.  MOVE 8 TO SS2.
    PERFORM MOVE-C  VARYING SS1 FROM 8 BY -1 UNTIL SS1 = 0.
MOVE-C.
    IF QTY-OR-COST (SS1) NOT = " " MOVE QTY-OR-COST (SS1) TO
    NC (SS2)  SUBTRACT 1 FROM SS2.
DISPLAY-NEW-STOCK.
    MOVE ON-HAND-QTY TO EDITED-QTY.
    COMPUTE  EDITED-COST = UNIT-COST / 100.
    COMPUTE  EDITED-VALUE = ON-HAND-QTY * UNIT-COST / 100.
    DISPLAY "NEW ON HAND QUANTITY = ", EDITED-QTY.
    DISPLAY "NEW UNIT COST = ", EDITED-COST.
    DISPLAY "NEW STOCK VALUE = ", EDITED-VALUE.
UPDATE-STOCK.
    COMPUTE UNIT-COST = (UNIT-COST * ON-HAND-QTY + NEW-QUANTITY
      * NEW-COST) / (ON-HAND-QTY + NEW-QUANTITY).
    COMPUTE ON-HAND-QTY = ON-HAND-QTY + NEW-QUANTITY.
    CALL "DBUPDATE" USING BASE-NAME, DATA-SET, MODE1, STATUSS,
      PREVIOUS-LIST, IP-BUFFER.
    IF CONDTN-WORD NOT = 0 DISPLAY "UPDATE FAIL"
      PERFORM DISPLAY-STATUS  GO TO FINISH.
    PERFORM UNLOCK.
UNLOCK.
    CALL "DBUNLOCK" USING BASE-NAME, DATA-SET, MODE1, STATUSS.
    IF CONDTN-WORD NOT = 0 DISPLAY "UNLOCK FAIL"
      PERFORM DISPLAY-STATUS  GO TO FINISH.
DISPLAY-STATUS.
    CALL "DBEXPLAIN" USING STATUSS.
FINISH.
    CALL "DBCLOSE" USING BASE-NAME, DATA-SET, MODE1, STATUSS.
    IF CONDTN-WORD  NOT = 0 DISPLAY
      "DATA BASE CLOSE FAILED"  PERFORM DISPLAY-STATUS.
    STOP  RUN.
```

Figure 5-1. Inventory Update Program (Continued)



```
:RUN RECEIVE

ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
4397D13P
NO SUCH STOCK NUMBER
ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
12345678
NO SUCH STOCK NUMBER
ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
6650DD2S
NO SUCH STOCK NUMBER
ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
6650D22S
NOW ENTER QUANTITY RECEIVED -
100

NOW ENTER UNIT COST IN CENTS -
150

NEW ON HAND QUANTITY =        306
NEW UNIT COST =               $2.44
NEW STOCK VALUE =           $746.63


ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
6650D22S
NOW ENTER QUANTITY RECEIVED -
5000

NOW ENTER UNIT COST IN CENTS -
1500

NEW ON HAND QUANTITY =      5306
NEW UNIT COST =             $14.27
NEW STOCK VALUE =        $75,716.62


ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
2457A11C
NOW ENTER QUANTITY RECEIVED -
10000000
NOW ENTER UNIT COST IN CENTS -
4000

NEW ON HAND QUANTITY =  11001345
NEW UNIT COST =               $50.31
NEW STOCK VALUE =    $553,477,666.95


ENTER 8 CHARACTER STOCK NUMBER OR TYPE SAYONARA
SAYONARA

 END OF PROGRAM
```

Figure 5-2. Sample RECEIVE Execution

FORTRAN

FORTRAN EXAMPLES

In the FORTRAN examples which follow, all variables are integer unless declared otherwise. The DUMMY parameter is an integer and appears when a parameter is not used by the procedure for the task which is being performed.

The code at statement 9900 closes the data base. It is not included in each example but is implied to be there.

Since IMAGE requires that the parameters be at word addresses, they must be integer arrays equivalenced to character strings if necessary. For example, the BASE integer array is 4 words long and is equivalenced to the CS1 character string which is 8 bytes long. This array contains the name of the STORE data base preceded by one word of blanks to which the data segment number of the DSCB is moved when the data base is opened.

OPEN DATA BASE

```
      PROGRAM FTIM1
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      CS1 = "  STORE;"
      CS2 = "        "
      DISPLAY "ENTER PASSWORD "
      ACCEPT CS2
      DISPLAY "ENTER ACCESS MODE (1-8) "
      ACCEPT MODE
      CALL DBOPEN (BASE,PASSWORD,MODE,STATUS)
      IF (STATUS(1).NE.0) GOTO 9300
      DISPLAY "DATA BASE OPENED"
      GOTO 9900
9300  DISPLAY "DBOPEN FAILURE"
9310  CALL DBEXPLAIN (STATUS)
                        .
                        .
                        .
```

In this example the STORE data base is opened in the access mode entered by the application user and with the user class number corresponding to the password entered. For example, the access mode may be 3 and the password DO-ALL. Since IMAGE parameters must have word addresses, the character string must be equivalenced to an integer array before being passed to the IMAGE procedure.

If the procedure fails, the first word of STATUS is an integer other than zero. In this case, the sample program prints a message and executes DBEXPLAIN to display status information.

If the password is less than 8 characters long, it must be followed by a semicolon or blank. Therefore, the character string CS2 is initialized to 8 blanks.

ADD ENTRY

```
      PROGRAM FTIM2
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETP(4),PRBUFF(14)
      CHARACTER*8 STOCKNO,CS3,DESCRIPN*20,ALLITEMS*2
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      EQUIVALENCE (DSETP(1),CS3), (PRBUFF(1),STOCKNO),
     C       (PRBUFF(5),DESCRIPN),(AI,ALLITEMS)
      CS1 = "  STORE;"
      CS2 = "        "
      CS3 = "PRODUCT;"
      MODE1=1
      ALLITEMS = "@;"
                           •
                           •
                           •
```

(code to open data base in access mode 1, 3, or 4 and prompt for data item values)

```
                           •
                           •
                           •
      CALL DBPUT (BASE,DSETP,MODE1,STATUS,AI,PRBUFF)
      IF (STATUS(1).NE.43) GOTO 120
      DISPLAY "DUPLICATE STOCK NUMBER"
      GOTO 110
  120 IF (STATUS(1).NE.16) GOTO 130
      DISPLAY "DATA SET FULL"
      GOTO 9900
  130 IF (STATUS(1).NE.-23) GOTO 140
      DISPLAY "PASSWORD DOES NOT ALLOW ADDING ENTRIES"
      GOTO 9900
  140 IF (STATUS(1).NE.0) GOTO 160
      DISPLAY "NEW PRODUCT HAS BEEN ENTERED"
      GOTO 9900
  160 DISPLAY "DBPUT FAILURE"
      GOTO 9310
 9300 DISPLAY "DBOPEN FAILURE"
 9310 CALL DBEXPLAIN (STATUS)
                           •
                           •
                           •
```

This sample code adds a data entry to the PRODUCT manual master data set. The first word of the BASE array now contains the number of the privileged data segment of the Data Base Control Block. ALLITEMS contains an at-sign indicating that PRBUFF contains a value for all items in the data entry. The values for the STOCK# and DESCRIPTION data items are concatenated in PRBUFF, for example, 7474Z74ZORANGE CRATEΔΔΔΔΔΔΔΔ.

A typical application will prompt for the data item values which are moved into PRBUFF and added to the data set. In this example, the condition word of the STATUS array is tested for a value of 43, indicating that an entry with search item value 7474Z74Z already exists in the data set, or 16, indicating that the data set is full. If the user's password does not allow entries to be added, condition word −23 is returned.

FORTRAN

If an entry is to be added to a detail set, a value must be provided for all search items and the sort item if one is defined. The program may first check to see if the required entries exist in the manual masters linked to the detail data set, or it can check for condition word 1*xx* after attempting to add the detail entry.

If the access mode is 1, the data base must be locked before an entry can be added.

READ ENTRY (SERIALLY)

```
      PROGRAM FTIM3
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETC(5),LIST(15),CBUFF(15)
      CHARACTER CS4*10,CS5*30,FNAME*10,LNAME*14
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (DSETC(1),CS4), (LIST(1),CS5), (CBUFF(3),FNAME),
     C        (CBUFF(8),LNAME)
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      CS1 = "  STORE;"
      CS2 = "        "
      CS4 = "CUSTOMER; "
      CS5 = "ACCOUNT,FIRST-NAME,LAST-NAME; "
      DUMMY = 1
      MODE2 = 2
          •
          •
          •
```

(code to open data base)

```
          •
          •
          •
  200 CALL DBGET (BASE,DSETC,MODE2,STATUS,LIST,CBUFF,DUMMY)
      IF (STATUS(1).NE.11) GOTO 210
      DISPLAY "CONTINUE"
      ACCEPT I
      IF (I.EQ.0) GOTO 9900
          •
          •
          •
```

(code to determine whether to continue, if so, rewind data set)

```
          •
          •
          •
  210 IF (STATUS(1).NE.-21.AND.STATUS(1).NE.-52) GOTO 220
      DISPLAY "YOU DO NOT HAVE ACCESS TO DATA"
      GOTO 9900
  220 IF (STATUS(1).EQ.0) GOTO 230
      DISPLAY "DBGET FAILURE"
      GOTO 9310
  230 WRITE (6,*) FNAME,LNAME,CBUFF(1)
      GOTO 200
 9300 DISPLAY "DBOPEN FAILURE"
 9310 CALL DBEXPLAIN (STATUS)
          •
          •
          •
```

To read the next entry of the CUSTOMER data set, a mode of 2 is used. This directs the DBGET procedure to perform a forward serial read. In the example, the LIST array contains the names of three data items. After DBGET returns to the calling program, CBUFF contains values such as:

CBUFF(1) – CBUFF(2)	12345678 (double integer)
CBUFF(3) – CBUFF(7)	GEORGEΔΔΔΔ
CBUFF(8) – CBUFF(14)	PADERSONΔΔΔΔΔΔ

If an end-of-file is encountered the condition word is set to 11. In this case, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the DBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read reads the first entry in the data set. If the user does not have read access to the data items, condition word –21 is returned.

The DUMMY variable signifies that the *argument* parameter is not used with mode 2.

READ ENTRY (DIRECTLY)

```
      PROGRAM FTIM11
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETP(4), PRBUFF(14)
      CHARACTER*8 CS1,CS2
      CHARACTER*8 CS3, STOCKNO, DESCRIPN*20, ALLITEMS*2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      EQUIVALENCE (DSETP(1),CS3), (PRBUFF(1),STOCKNO),
     C        (PRBUFF(5),DESCRIPN), (AI,ALLITEMS)
      INTEGER*4 RECNO
      CS1 = "  STORE;"
      CS2 = "        "
      CS3 = "PRODUCT;"
      ALLITEMS = "@;"
      MODE4 = 4
           •
           •
```

(code to open data base)

```
           •
           •
  210 DISPLAY "REC"
      ACCEPT RECNO
      IF (RECNO.EQ.0) GOTO 9900
      CALL DBGET (BASE, DSETP, MODE4, STATUS, AI, PRBUFF, RECNO)
      IF (STATUS(1).EQ.12.OR.STATUS(1).EQ.13) GOTO 280
      IF (STATUS(1).NE.17) GOTO 270
      DISPLAY "RECORD CONTAINS NO DATA ENTRY"
      GOTO 210
  270 IF (STATUS(1).EQ.0) GOTO 290
      DISPLAY "DBGET FAILURE"
      GOTO 9310
  280 DISPLAY "INCORRECT RECORD NUMBER"
      GOTO 210
  290 DISPLAY DESCRIPN
      GOTO 210
 9300 DISPLAY "DBOPEN FAILURE"
 9310 CALL DBEXPLAIN (STATUS)
           •
           •
```

FORTRAN

The code in this example reads all data items of the entry in the specified record number of the PRODUCT data set using a directed read, mode 4. If the condition word is equal to 12 or 13, the record number is not within the range of records in the file. If the condition word is 17 the record contains no entry.

> NOTE
>
> This is not the normal method for using directed reads but is used to simplify the example.

READ ENTRY (CALCULATED)

```
      PROGRAM FTIM4
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETP(4),LISTA(6),PRBUFF(10),STOCKSRCH(4)
      CHARACTER*8 CS1,CS2
      CHARACTER*8 CS3, DESCRIPN*20, CS6*12, CS7
      EQUIVALENCE (DSETP(1),CS3),(PRBUFF(1),DESCRIPN),
     C       (LISTA(1),CS6),(STOCKSRCH(1),CS7)
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      CS1 = "  STORE;"
      CS2 = "        "
      CS3 = "PRODUCT;"
      CS6 = "DESCRIPTION;"
      MODE7 = 7
          •
          •
          •
(code to open data base)

          •
          •
          •

   20 DISPLAY "STOCK NUMBER"
      ACCEPT CS7
      CALL DBGET (BASE,DSETP,MODE7,STATUS,LISTA,PRBUFF(1),STOCKSRCH)
      IF (STATUS(1).NE.17) GOTO 300
      DISPLAY "NO SUCH STOCK NUMBER"
      GOTO 20
  300 IF (STATUS(1).NE.-21) GOTO 310
      DISPLAY "PASSWORD DOES NOT GRANT ACCESS TO DATA"
      GOTO 9900
  310 IF (STATUS(1).EQ.0) GOTO 320
      DISPLAY "DBGET FAILURE"
      GOTO 9310
  320 (code to use data from entry just read)

          •
          •
          •

 9310 CALL DBEXPLAIN (STATUS)
          •
          •
          •
```

A calculated read is used to locate the PRODUCT data set entry which has the STOCK# search item value entered in CS7. The mode is 7 and the item to be read is DESCRIPTION. After DBGET returns control to the calling program, the description for the specified stock number is in DESCRIPN. If no entry exists with STOCK# equal to the specified value, the condition word is 17. If the user does not have read access to the DESCRIPTION data item, the condition word is –21.

READ ENTRY (FORWARD CHAIN)

```
      PROGRAM FTIM5
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETS(3), INAME(6), IVAL(3), SABUFF(19)
      CHARACTER CS8*6, CS9*12, CS10*6, SASTOCK*8, PURCHDT*8,
     C                  DELIVDT*8, ALLITEMS*2
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      EQUIVALENCE (DSETS(1),CS8), (INAME(1),CS9), (IVAL(1),CS10),
     C          (SABUFF(1),ACCTS), (SABUFF(3),SASTOCK),
     C          (SABUFF(7),QTY), (SABUFF(8),PRICE),
     C          (SABUFF(10),TAX), (SABUFF(12),TOTAL),
     C          (SABUFF(14),PURCHDT), (SABUFF(17),DELIVDT),(AI,ALLITEMS)
      INTEGER*4 ACCTS, PRICE, TAX, TOTAL
      CS1 = "  STORE;"
      CS2 = "        "
      CS8 = "SALES;"
      CS9 = "PURCH-DATE; "
      CS10 = "760314"←———————— Program would normally prompt for this value.
      ALL ITEMS = "@;"
      MODE1 = 1
      MODE5 = 5
         •
         •
(code to open data base)
         •
      CALL DBFIND (BASE, DSETS, MODE1, STATUS, INAME, IVAL)
      IF (STATUS(1).NE.17) GOTO 345
      DISPLAY "NO PURCHASES ON THAT DATE."
      GOTO 9900
  345 IF (STATUS(1).NE.-21.AND.STATUS(1).NE.-52) GOTO 355
      DISPLAY "PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS"
      GOTO 9900
  355 IF (STATUS(1).EQ.0) GOTO 360
      DISPLAY "DBFIND FAILURE"
      GOTO 9310
  360 CALL DBGET (BASE, DSETS, MODE5, STATUS, AI, SABUFF, DUMMY)
      IF (STATUS(1).NE.15) GOTO 365
      DISPLAY "NO MORE PURCHASES ON THIS DATE"
      GOTO 9900
  365 IF (STATUS(1).NE.0) GOTO 380
```

(code to use sales information from entry, for example, in a report)

```
      GOTO 360
  380 DISPLAY "DBGET FAILURE"
      GOTO 9310
 9300 DISPLAY "DBOPEN FAILURE"
 9310 CALL DBEXPLAIN (STATUS)
```

FORTRAN

First the DBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value 760314 of both the master entry containing the chain head and the detail entries making up the chain. If no entry in the DATE-MASTER has a search item value of 760314, the condition word will be 17. If the user's password or access mode does not grant read access to the data set or data items, condition word −21 or −52 is returned.

If the DBFIND procedure executes successfully, a call to the DBGET procedure with a mode parameter of 5 reads the first entry in the chain if one exists. Subsequent calls to DBGET with the same mode read the succeeding entries to the chain until the last entry in the chain has been read. If the condition word is 15, the end of the chain has been reached and no more entries are available, or no entries exist in the chain.

If an entry is successfully read the program uses the information and then returns to statement 360 to read another entry in the chain.

After an entry has been read the SABUFF array contains information like this:

SABUFF (1)	— SABUFF(2)	ACCTS	12345678	(doubleword integer)
SABUFF(3)	— SABUFF(6)	SASTOCK	2222B22B	(character string)
SABUFF(7)		QTY	3	(integer)
SABUFF(8)	— SABUFF(9)	PRICE	425	(doubleword integer)
SABUFF(10)	— SABUFF(11)	TAX	25	(doubleword integer)
SABUFF(12)	— SABUFF(13)	TOTAL	450	(doubleword integer)
SABUFF(14)	— SABUFF(16)	PURCHDT	760314	(character string)
SABUFF(17)	— SABUFF(19)	DELIVDT	760320	(character string)

UPDATE ENTRY

```
      PROGRAM FTIM6
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETC(5), INAME2(8), ADDVAL(13)
      INTEGER*4 ACCTSRCH
      CHARACTER CS4*10, CS11*16, ADDSTRING*26
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      EQUIVALENCE (DSETC(1),CS4), (INAME2(1),CS11),
     C                 (ADDVAL(1),ADDSTRING)
      CS1 = "  STORE;"
      CS2 = "        "
      CS4 = "CUSTOMER; "
      CS11 = "STREET-ADDRESS; "
      MODE1 = 1
      MODE7 = 7
      ACCTSRCH = 12345678
                            •
                            •
```

(code to open data base in access mode 1, 2, 3, or 4)

```
                            •
                            •
      CALL DBGET (BASE,DSETC,MODE7,STATUS,INAME2,ADDVAL,ACCTSRCH)
                            •
```

(code to determine if read is successful and print current address)

```
                            •
      DISPLAY "NEW ADDRESS"
      ACCEPT ADDSTRING
      CALL DBUPDATE (BASE, DSETC, MODE1, STATUS, INAME2, ADDVAL)
      IF (STATUS(1).NE.42) GOTO 420
      DISPLAY "YOU ARE NOT ALLOWED TO ALTER THIS ITEM"
      GOTO 9900
  420 IF (STATUS(1).EQ.0) GOTO 440
      DISPLAY "DBUPDATE FAILURE"
      GOTO 9310
  440 DISPLAY "ADDRESS CHANGED"
      GOTO 9900
 9300 DISPLAY "DBOPEN FAILURE"
 9310 CALL DBEXPLAIN (STATUS)
```

Before an entry can be updated it must be located. In this example, the entry is located by using a calculated DBGET to read the STREET-ADDRESS item in the CUSTOMER data set. The entry is located by using the ACCOUNT search item with a value of 12345678. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into ADDRESS-VALUE. The DBUPDATE routine is then called to alter the STREET-ADDRESS data item in the entry.

If the current user class number does not allow this item to be altered or the access mode does not allow updates to take place, the condition word 42 is returned.

A null list can be used with DBGET to locate an entry to be updated.

FORTRAN

DELETE ENTRY

```
      PROGRAM FTIM7
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETC(5), INAME2(8), ADDVAL(13)
      INTEGER*4 ACCTSRCH
      CHARACTER CS4*10, CS11*16, ADDSTRING*26
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      EQUIVALENCE (DSETC(1),CS4), (INAME2(1),CS11),
     C                  (ADDVAL(1),ADDSTRING)
      CS1 = "  STORE;"
      CS2 = "        "
      CS4 = "CUSTOMER; "
      CS11 = "; "
      MODE1 = 1
      MODE7 = 7
                          •
                          •
```

(code to open data base in access mode 1, 3, or 4)

```
                          •
                          •

   20 DISPLAY "ACCOUNT OR ZERO TO TERMINATE"
      ACCEPT ACCTSRCH
      IF (ACCTSRCH.EQ.0) GOTO 9900
      CALL DBGET (BASE,DSETC,MODE7,STATUS,INAME2,DUMMY,ACCTSRCH)
      IF (STATUS(1).NE.0) GOTO 9310
      CALL DBDELETE (BASE, DSETC, MODE1, STATUS)
      IF (STATUS(1).NE.44) GOTO 530
      DISPLAY "SALES ENTRIES EXIST, CUSTOMER CANNOT BE DELETED"
      GOTO 20
  530 IF (STATUS(1).NE.-23) GOTO 540
      DISPLAY "PASSWORD DOES NOT GRANT ACCESS TO DATA SET"
      GOTO 9900
  540 IF (STATUS(1).EQ.0) GOTO 560
      DISPLAY "DBDELETE FAILURE"
      GOTO 9310
  560 DISPLAY "CUSTOMER ENTRY DELETED"
      GOTO 20
 9300 DISPLAY "DBOPEN FAILURE"
 9310 CALL DBEXPLAIN (STATUS)
                          •
                          •
                          •
```

Before an entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling DBGET. In this example, the program may have requested the account number of the customer to be deleted and then used a calculated DBGET to locate the appropriate entry. If entries in the SALES data set exist which have the same account number as the entry to be deleted, the condition word is set to 44 and the entry is not deleted.

A null list can be used with DBGET to locate an entry to be deleted.

If the access mode is 1, the data base must be locked before the entry is deleted.

LOCK AND UNLOCK (DATA BASE)

```
      PROGRAM FTIM8
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      CS1 = "  STORE;"
      CS2 = "        "
                                 •
                                 •
                                 •
```

(code to open data base in access mode 1 or 5)

```
                                 •
                                 •
                                 •
      MODE1 = 1
      MODE2 = 2
      CALL DBLOCK (BASE, DUMMY, MODE2, STATUS)
      IF (STATUS(1).NE.20) GOTO 640
      DISPLAY "DATA BASE IS BUSY. TRY AGAIN LATER."
      GOTO 9900
640   IF (STATUS(1).EQ.0) GOTO 680
      DISPLAY "DBLOCK FAILURE"
      GOTO 9310
680     (code to use data base)
                                 •
                                 •
                                 •
      CALL DBUNLOCK (BASE, DUMMY, MODE1, STATUS)
      IF (STATUS(1).EQ.0) GOTO 9900
      DISPLAY "DBUNLOCK FAILURE"
      GOTO 9310
9300  DISPLAY "DBOPEN FAILURE"
9310  CALL DBEXPLAIN (STATUS)
                                 •
                                 •
                                 •
```

In this example, the program calls DBLOCK to lock the data base. Since mode 2 is used, the program must check the condition word when DBLOCK returns control to verify that the data base is locked and the calling program has exclusive access. If this is so, the condition word is 0; if it is busy the condition word is 20.

If the data base is successfully locked, the program performs the necessary data base operations and then unlocks the data base by calling the DBUNLOCK procedure. In the example the programs terminates after unlocking the data base.

LOCK (DATA ENTRIES)

```
      PROGRAM FTIM8A
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4),IP(40)
      CHARACTER*8 CS1,CS2,VAL
      CHARACTER*2 RELOP
      CHARACTER*16 SETNAME,ITEMNAME
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2),
     +            (NUM,IP(1)),
     +            (LENGTH,IP(2)),
     +            (SETNAME,IP(3)),
     +            (ITEMNAME,IP(11)),
     +            (RELOP,IP(19)),
     +            (VAL,IP(20))
      CS1 = "  STORE;"
      CS2 = "        "
      MODE1=1
                •
                •
  (code to open data base)
                •
                •
      NUM=1
      LENGTH=22
      SETNAME="INVENTORY       "
      ITEMNAME="STOCK#          "
      RELOP="= "
      VAL="6650D22S"
      CALL DBLOCK (BASE,IP,5,STATUS)
  640 IF (STATUS(1).EQ.0) GOTO 680
      DISPLAY "DBLOCK FAILURE"
      GOTO 9310
  680        (code to modify the locked data entry or entries)
                •
                •
                •
 9310 CALL DBEXPLAIN (STATUS)
                •
                •
```

This example illustrates locking at the data entry level. All data entries in the INVENTORY data set with a STOCK# value of 6650D22S are locked unconditionally (mode 5). If the lock request succeeds, the condition word is 0. If the DBLOCK procedure detects a calling error or an exceptional condition such as DBCB full, the DBLOCK failure message is displayed and DBEXPLAIN is called.

REQUEST DATA SET INFORMATION

```
      PROGRAM FTIM9
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION INFOBUF(8)
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      CS1 = "  STORE;"
      CS2 = "        "
                  •
                  •
                  •
  (code to open data base)
                  •
                  •
                  •
      MODE=203
      CALL DBINFO (BASE, DUMMY, MODE, STATUS, INFOBUF)
      IF (STATUS(1).EQ.0) GOTO 700
      DISPLAY "DBINFO FAILURE"
      GOTO 9310
 700  (code to use data set numbers returned in INFOBUF)
                  •
                  •
                  •
      GOTO 9900
9300  DISPLAY "DBOPEN FAILURE"
9310  CALL DBEXPLAIN (STATUS)
                  •
                  •
                  •
```

The procedure call in this example obtains the numbers of the data sets available to the current user class by specifying mode 203. If the user class number is 12, after the call has been successfully executed the INFOBUF array contains:

INFOBUF(1)	4	Access to 4 data sets.
INFOBUF(2)	2	Read access to data set 2.
INFOBUF(3)	–3	Modify access to data set 3
INFOBUF(4)	–5	and data set 5.
INFOBUF(5)	6	Read and possibly update access to data set 6.

If the user class number is 8 it contains:

INFOBUF(1)	6	Access to 6 data sets.
INFOBUF(2)	–1	Modify access to data set 1.
INFOBUF(3)	2	Read access to data set 2, an automatic master.
INFOBUF(4)	–3	Modify access to all the other data sets.
INFOBUF(5)	–4	
INFOBUF(6)	–5	
INFOBUF(7)	–6	

Refer to the schema in figure 3-5 to help you interpret this procedure call in relation to the STORE data base.

REWIND DATA SET

```
      PROGRAM FTIM3
      IMPLICIT INTEGER (A-Z)
      DIMENSION STATUS(10),PASSWORD(4),BASE(4)
      DIMENSION DSETC(5),LIST(15),CBUFF(15)
      CHARACTER CS4*10,CS5*30,FNAME*10,LNAME*14
      CHARACTER*8 CS1,CS2
      EQUIVALENCE (DSETC(1),CS4), (LIST(1),CS5), (CBUFF(3),FNAME),
     C        (CBUFF(8),LNAME)
      EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
      CS1 = "  STORE;"
      CS2 = "        "
      CS4 = "CUSTOMER; "
                  •
                  •
                  •
      MODE3 = 3
      CALL DBCLOSE (BASE,DSETC,MODE3,STATUS)
      IF (STATUS(1).EQ.0) GOTO 200
      DISPLAY "DBCLOSE FAILURE"
      GOTO 9310
```

To rewind the CUSTOMER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the Data Set Control Block for CUSTOMER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request will read the first entry in the data set.

CLOSE DATA BASE

```
9900  MODE1=1
      CALL DBCLOSE (BASE,DUMMY,MODE1,STATUS)
      IF (STATUS(1).EQ.0) GOTO 9980
      DISPLAY "DBCLOSE FAILURE"
      CALL DBEXPLAIN (STATUS)
9980  STOP
      END
```

This call closes the data base. It is issued after the program has completed all data base operations and before program termination.

PRINT ERROR

```
               •
               •
               •
     CALL DBEXPLAIN (STATUS)
9980 STOP
     END
```

A call to DBEXPLAIN prints a message on the $STDLIST device which interprets the contents of the STATUS array.

MOVE ERROR TO BUFFER

```
     PROGRAM FTIM10
     IMPLICIT INTEGER (A-Z)
     DIMENSION STATUS(10),PASSWORD(4),BASE(4)
     DIMENSION ERBUFF(36)
     CHARACTER ERSTRING*72
     CHARACTER*8 CS1,CS2
     EQUIVALENCE (BASE(1),CS1),(PASSWORD(1),CS2)
     EQUIVALENCE (ERBUFF(1),ERSTRING)
     CS1 = "  STORE;"
     CS2 = "        "
     DISPLAY "ENTER PASSWORD "
     ACCEPT CS2
     DISPLAY "ENTER ACCESS MODE (1-8) "
     ACCEPT MODE
     CALL DBOPEN (BASE,PASSWORD,MODE,STATUS)
     IF (STATUS(1).NE.0) GOTO 9300
               •
               •
               •
9300 DISPLAY "DBOPEN FAILURE"
9310 CALL DBERROR (STATUS,ERBUFF,LENG)
     DISPLAY ERSTRING [1:LENG]
               •
               •
```

In this example, a call to DBERROR returns one of the messages appropriate to the condition word returned by the DBOPEN procedure if it fails. For example, the message in ERSTRING may be DATA BASE OPEN IN AN INCOMPATIBLE MODE if the condition word is –1. The value of LENG in this case is 38.

SPL EXAMPLES

Figures 5-3 and 5-5 illustrate the use of IMAGE procedures with SPL programs. The program in figure 5-3 is called SUPPLMOD. It opens the data base in access mode 1 and allows the user to update, add, and delete entries of the master data set containing information on suppliers. Sample output from SUPPLMOD is illustrated in figure 5-4.

Figure 5-5 contains a program called SHOWSALE, which displays credit card purchase transactions from the detail data set containing these entries. SHOWSALE opens the data base in access mode 6 thereby avoiding the necessity of locking and unlocking the data base. Figure 5-6 shows the output from SHOWSALE.

(1) IMAGE procedures must be named in an INTRINSIC statement or, alternatively, declared as EXTERNAL procedures.

(2) OPEN DATA BASE

The STORE data base is opened with access mode 1 and BUYER password. If the condition code is not zero, an error message is printed and the program terminates. (See (13) for another example of opening a data base.)

(3) MOVE ERROR TO BUFFER

A message explaining the condition word returned by DBOPEN is moved to OUTBUF and I is set equal to the number of characters or length of the message.

(4) REQUEST DATA SET INFORMATION

A call to DBINFO with mode 201 and data set name SUP-MASTER returns the data set number in DSET. For efficiency, this is done only once at the beginning of the program. (See (14) and (16) for other DBINFO examples.)

(5) LOCK DATA SET

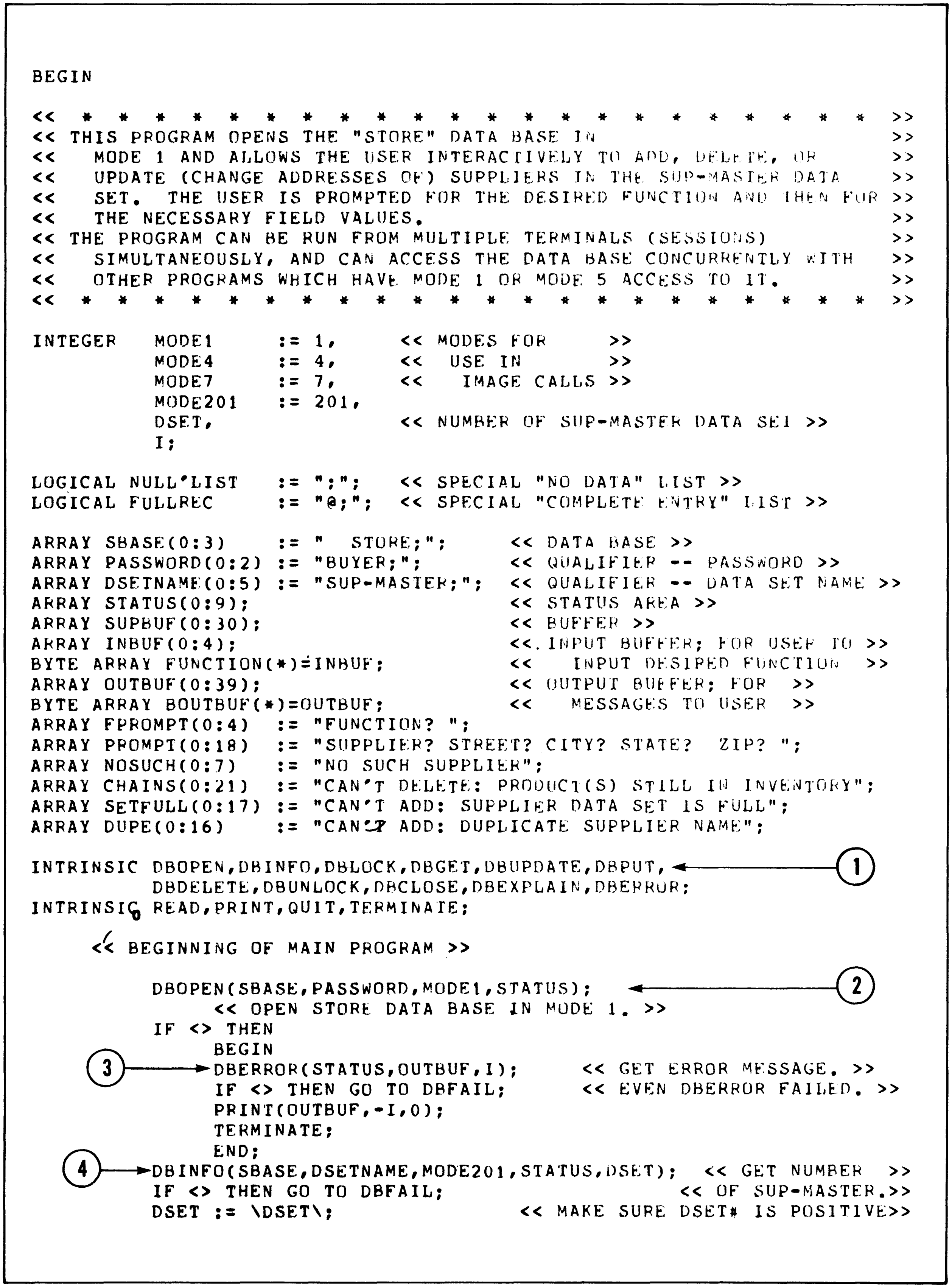
If the condition code is not CCE, the status information is printed indicating that the lock could not be obtained.

(6) READ ENTRY (CALCULATED)

This call locates an entry in the SUP-MASTER data set based on the search item value in SUPBUF. The entry need not be read since it is to be updated or deleted, therefore, the list is null and no data is actually transferred. SUPBUF is also used as the buffer parameter since no data is moved into it. If the condition word is 17, there is no search item with the specified value. Since the BUYER password allows access to the data, it is not necessary to check for condition word -21.

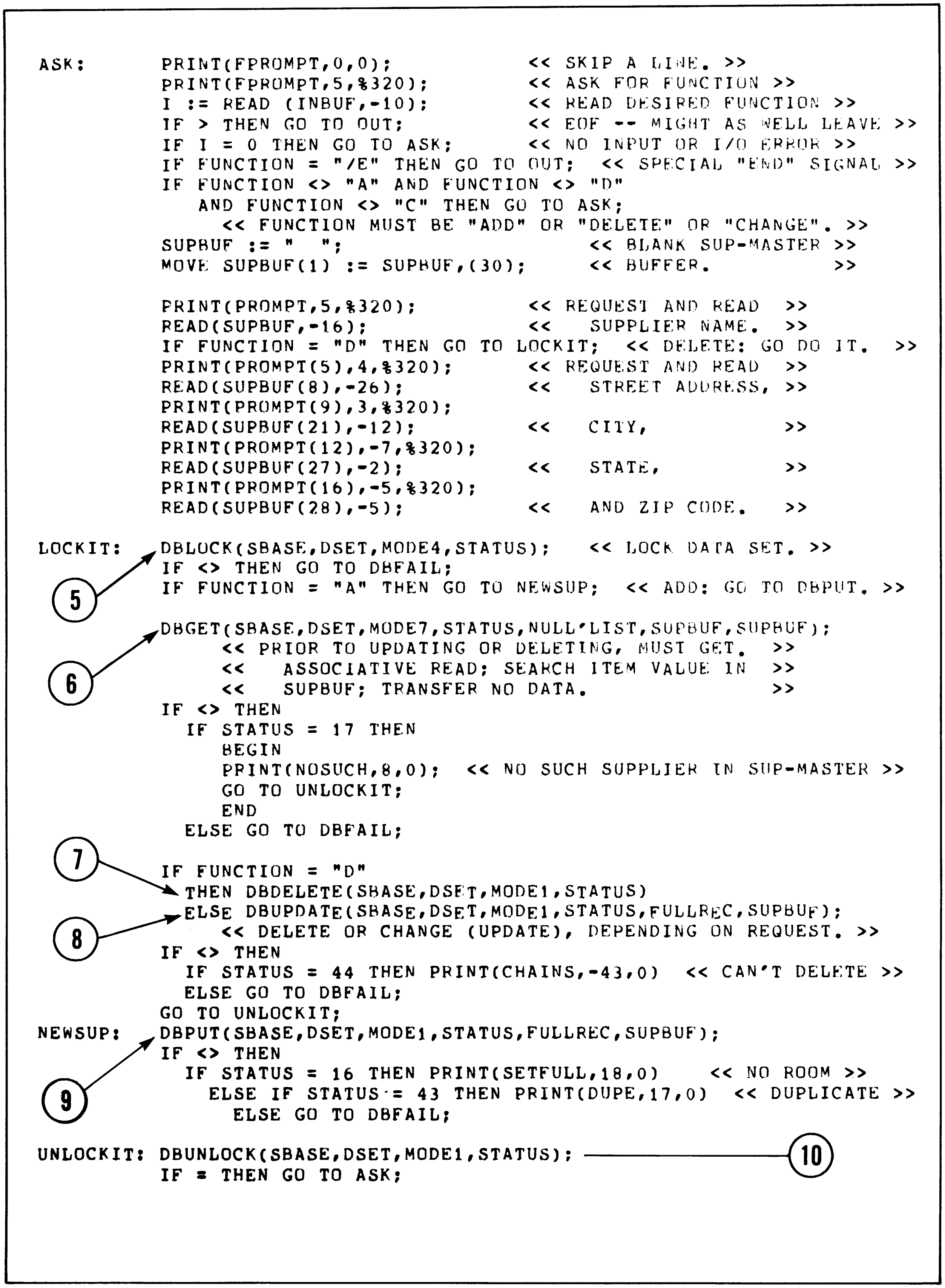
(7) DELETE ENTRY

The entry located with DBGET is deleted. If the condition word is 44, the detail data sets linked to SUP-MASTER contain entries with the specified search item value, therefore, the master entry cannot be deleted. Since the BUYER password and access mode 1 allow the user to delete SUP-MASTER entries, it is not necessary to check for condition word -23.



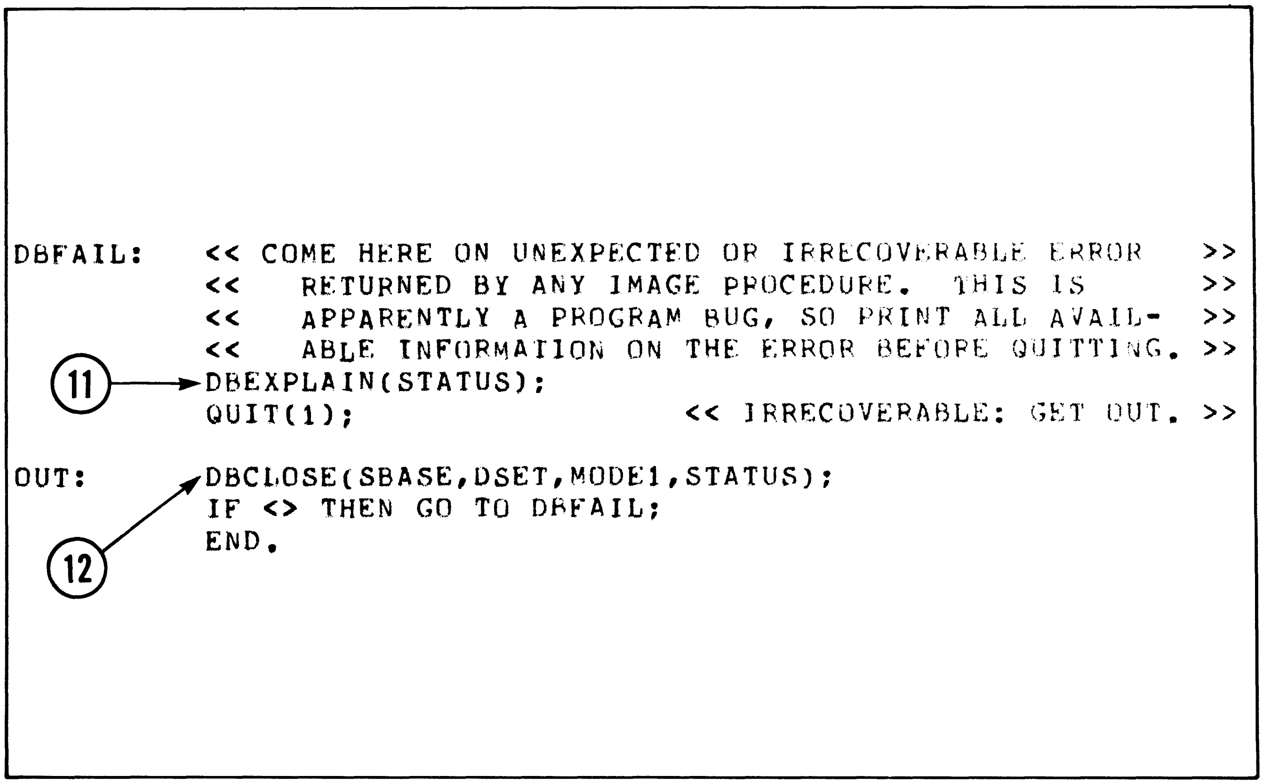
```
BEGIN

<<  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *  >>
<< THIS PROGRAM OPENS THE "STORE" DATA BASE IN                                             >>
<<    MODE 1 AND ALLOWS THE USER INTERACTIVELY TO ADD, DELETE, OR                          >>
<<    UPDATE (CHANGE ADDRESSES OF) SUPPLIERS IN THE SUP-MASTER DATA                        >>
<<    SET.  THE USER IS PROMPTED FOR THE DESIRED FUNCTION AND THEN FOR                     >>
<<    THE NECESSARY FIELD VALUES.                                                          >>
<< THE PROGRAM CAN BE RUN FROM MULTIPLE TERMINALS (SESSIONS)                               >>
<<    SIMULTANEOUSLY, AND CAN ACCESS THE DATA BASE CONCURRENTLY WITH                       >>
<<    OTHER PROGRAMS WHICH HAVE MODE 1 OR MODE 5 ACCESS TO IT.                             >>
<<  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *  >>

INTEGER   MODE1      := 1,     << MODES FOR      >>
          MODE4      := 4,     <<  USE IN        >>
          MODE7      := 7,     <<   IMAGE CALLS >>
          MODE201    := 201,
          DSET,                << NUMBER OF SUP-MASTER DATA SET >>
          I;

LOGICAL NULL'LIST    := ";";    << SPECIAL "NO DATA" LIST >>
LOGICAL FULLREC      := "@;";   << SPECIAL "COMPLETE ENTRY" LIST >>

ARRAY SBASE(0:3)     := "  STORE;";       << DATA BASE >>
ARRAY PASSWORD(0:2)  := "BUYER;";         << QUALIFIER -- PASSWORD >>
ARRAY DSETNAME(0:5)  := "SUP-MASTER;";    << QUALIFIER -- DATA SET NAME >>
ARRAY STATUS(0:9);                        << STATUS AREA >>
ARRAY SUPBUF(0:30);                       << BUFFER >>
ARRAY INBUF(0:4);                         << INPUT BUFFER; FOR USER TO >>
BYTE ARRAY FUNCTION(*)=INBUF;             <<   INPUT DESIRED FUNCTION  >>
ARRAY OUTBUF(0:39);                       << OUTPUT BUFFER; FOR  >>
BYTE ARRAY BOUTBUF(*)=OUTBUF;             <<   MESSAGES TO USER  >>
ARRAY FPROMPT(0:4)   := "FUNCTION? ";
ARRAY PROMPT(0:18)   := "SUPPLIER? STREET? CITY? STATE?  ZIP? ";
ARRAY NOSUCH(0:7)    := "NO SUCH SUPPLIER";
ARRAY CHAINS(0:21)   := "CAN'T DELETE: PRODUCT(S) STILL IN INVENTORY";
ARRAY SETFULL(0:17)  := "CAN'T ADD: SUPPLIER DATA SET IS FULL";
ARRAY DUPE(0:16)     := "CAN'T ADD: DUPLICATE SUPPLIER NAME";

INTRINSIC DBOPEN,DBINFO,DBLOCK,DBGET,DBUPDATE,DBPUT, <──────────── 1
          DBDELETE,DBUNLOCK,DBCLOSE,DBEXPLAIN,DBERROR;
INTRINSIC READ,PRINT,QUIT,TERMINATE;

     << BEGINNING OF MAIN PROGRAM >>

          DBOPEN(SBASE,PASSWORD,MODE1,STATUS);  <──────────── 2
               << OPEN STORE DATA BASE IN MODE 1. >>
          IF <> THEN
               BEGIN
 3 ───────────>DBERROR(STATUS,OUTBUF,I);      << GET ERROR MESSAGE. >>
               IF <> THEN GO TO DBFAIL;      << EVEN DBERROR FAILED. >>
               PRINT(OUTBUF,-I,0);
               TERMINATE;
               END;
 4 ──────>DBINFO(SBASE,DSETNAME,MODE201,STATUS,DSET);  << GET NUMBER  >>
          IF <> THEN GO TO DBFAIL;                   << OF SUP-MASTER.>>
          DSET := \DSET\;                  << MAKE SURE DSET# IS POSITIVE>>
```

Figure 5-3. Supplier Modification Program



```
ASK:       PRINT(FPROMPT,0,0);            << SKIP A LINE. >>
           PRINT(FPROMPT,5,%320);         << ASK FOR FUNCTION >>
           I := READ (INBUF,-10);         << READ DESIRED FUNCTION >>
           IF > THEN GO TO OUT;           << EOF -- MIGHT AS WELL LEAVE >>
           IF I = 0 THEN GO TO ASK;       << NO INPUT OR I/O ERROR >>
           IF FUNCTION = "/E" THEN GO TO OUT;  << SPECIAL "END" SIGNAL >>
           IF FUNCTION <> "A" AND FUNCTION <> "D"
              AND FUNCTION <> "C" THEN GO TO ASK;
              << FUNCTION MUST BE "ADD" OR "DELETE" OR "CHANGE". >>
           SUPBUF := "  ";                     << BLANK SUP-MASTER >>
           MOVE SUPBUF(1) := SUPBUF,(30);      << BUFFER.          >>

           PRINT(PROMPT,5,%320);          << REQUEST AND READ  >>
           READ(SUPBUF,-16);              <<   SUPPLIER NAME.  >>
           IF FUNCTION = "D" THEN GO TO LOCKIT;  << DELETE: GO DO IT.  >>
           PRINT(PROMPT(5),4,%320);       << REQUEST AND READ  >>
           READ(SUPBUF(8),-26);           <<   STREET ADDRESS, >>
           PRINT(PROMPT(9),3,%320);
           READ(SUPBUF(21),-12);          <<   CITY,           >>
           PRINT(PROMPT(12),-7,%320);
           READ(SUPBUF(27),-2);           <<   STATE,          >>
           PRINT(PROMPT(16),-5,%320);
           READ(SUPBUF(28),-5);           <<   AND ZIP CODE.   >>

LOCKIT:    DBLOCK(SBASE,DSET,MODE4,STATUS);    << LOCK DATA SET. >>
           IF <> THEN GO TO DBFAIL;
           IF FUNCTION = "A" THEN GO TO NEWSUP;  << ADD: GO TO DBPUT. >>

           DBGET(SBASE,DSET,MODE7,STATUS,NULL'LIST,SUPBUF,SUPBUF);
                << PRIOR TO UPDATING OR DELETING, MUST GET.  >>
                <<   ASSOCIATIVE READ; SEARCH ITEM VALUE IN  >>
                <<   SUPBUF; TRANSFER NO DATA.               >>
           IF <> THEN
             IF STATUS = 17 THEN
                BEGIN
                PRINT(NOSUCH,8,0);  << NO SUCH SUPPLIER IN SUP-MASTER >>
                GO TO UNLOCKIT;
                END
             ELSE GO TO DBFAIL;

           IF FUNCTION = "D"
             THEN DBDELETE(SBASE,DSET,MODE1,STATUS)
             ELSE DBUPDATE(SBASE,DSET,MODE1,STATUS,FULLREC,SUPBUF);
                << DELETE OR CHANGE (UPDATE), DEPENDING ON REQUEST. >>
           IF <> THEN
             IF STATUS = 44 THEN PRINT(CHAINS,-43,0)  << CAN'T DELETE >>
             ELSE GO TO DBFAIL;
           GO TO UNLOCKIT;
NEWSUP:    DBPUT(SBASE,DSET,MODE1,STATUS,FULLREC,SUPBUF);
           IF <> THEN
             IF STATUS = 16 THEN PRINT(SETFULL,18,0)    << NO ROOM >>
               ELSE IF STATUS = 43 THEN PRINT(DUPE,17,0)  << DUPLICATE >>
                 ELSE GO TO DBFAIL;

UNLOCKIT:  DBUNLOCK(SBASE,DSET,MODE1,STATUS);
           IF = THEN GO TO ASK;
```

Figure 5-3. Supplier Modification Program (Continued)



```
DBFAIL:     << COME HERE ON UNEXPECTED OR IRRECOVERABLE ERROR      >>
            <<    RETURNED BY ANY IMAGE PROCEDURE.  THIS IS        >>
            <<    APPARENTLY A PROGRAM BUG, SO PRINT ALL AVAIL-    >>
            <<    ABLE INFORMATION ON THE ERROR BEFORE QUITTING.   >>
            DBEXPLAIN(STATUS);
            QUIT(1);                 << IRRECOVERABLE: GET OUT.    >>

OUT:        DBCLOSE(SBASE,DSET,MODE1,STATUS);
            IF <> THEN GO TO DBFAIL;
            END.
```

Figure 5-3. Supplier Modification Program (Continued)

⑧ UPDATE ENTRY

The entry located with DBGET is updated with the data in SUPBUF. The user is prompted for this data prior to the call to DBGET. FULLREC contains @; which indicates the entire entry is to be updated. In this case, the search item value must equal the value that is already in the entry. Since the BUYER password and access mode 1 allow updates to this data set, it is not necessary to check for condition word 42.

⑨ ADD ENTRY

This call adds an entry to the SUP-MASTER data set using the data in SUPBUF. The list parameter in FULLREC is @; specifying that values are provided for all data items in the entry. If the condition word is 16, the data set is full and, if it is 43, there is already an entry with the specified search item value. Since the BUYER password and access mode 1 allow adding entries to the data set, it is not necessary to check for condition word –23.

⑩ UNLOCK DATA SET

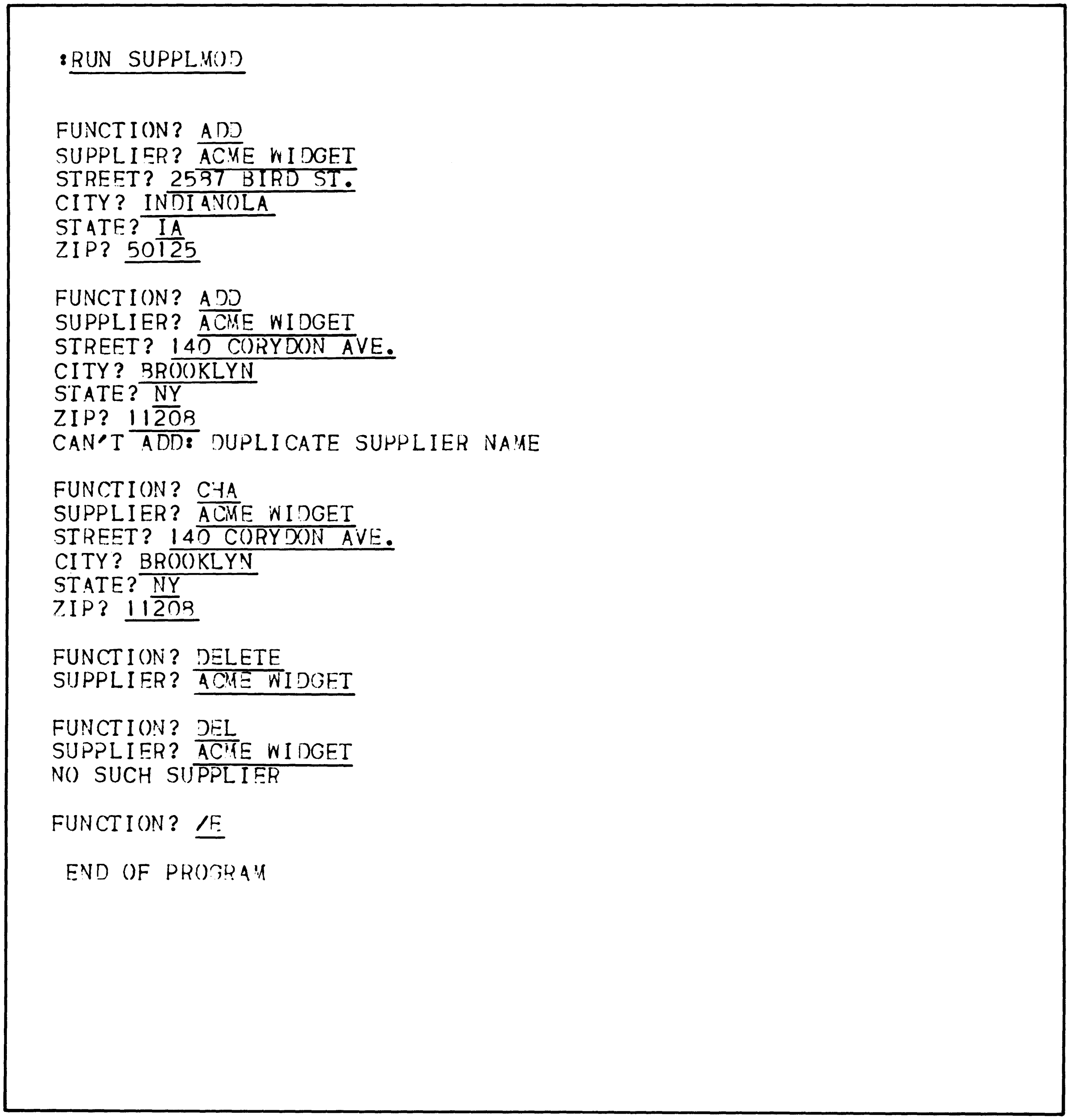
This call unlocks the data set. The DSET parameter is ignored. If the call is successful, the condition code is CCE and the program branches to ASK.

⑪ PRINT ERROR

A call to DBEXPLAIN prints a message interpreting the STATUS array contents.

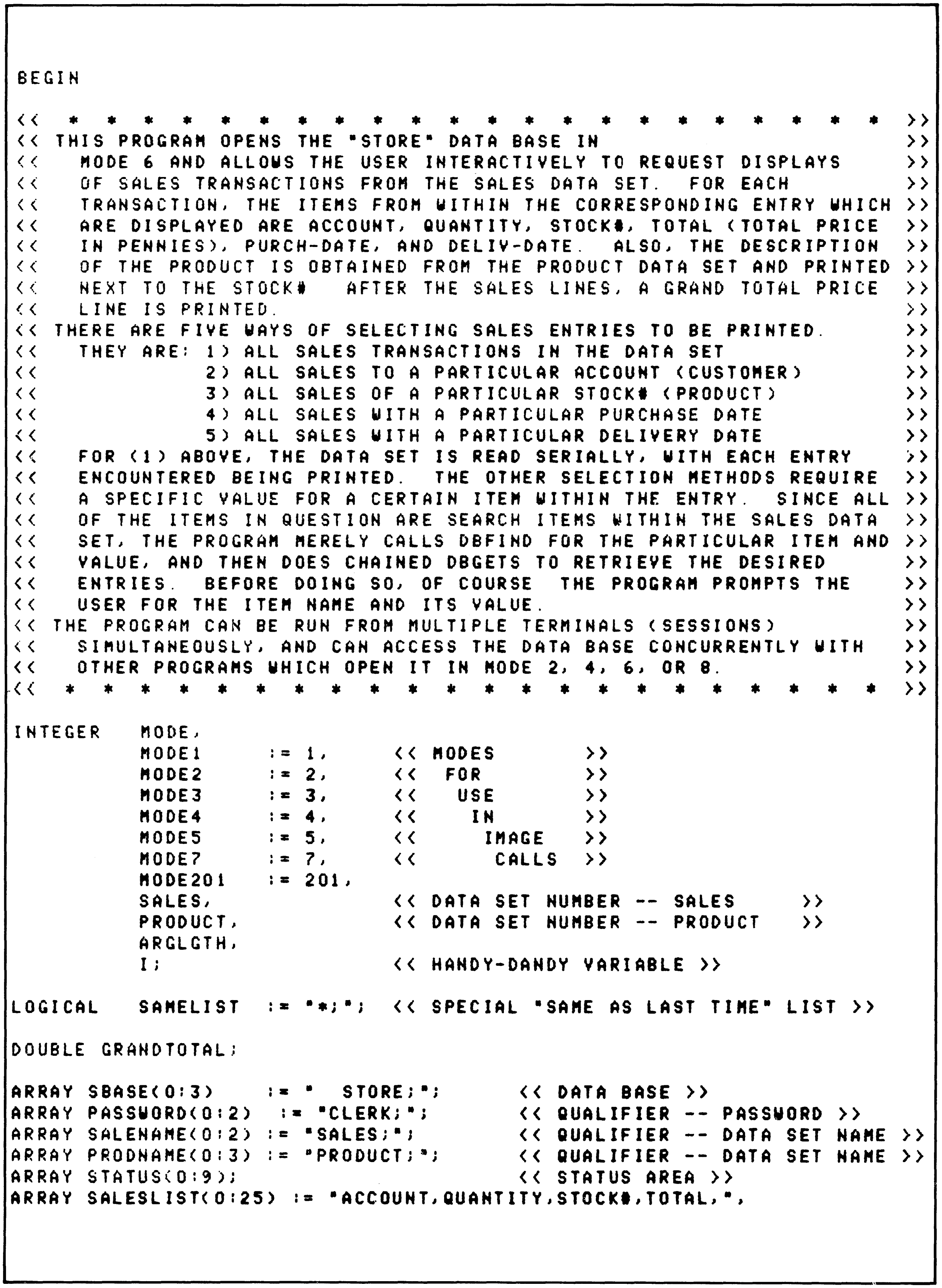
⑫ CLOSE DATA BASE

This call closes the STORE data base. The DSET parameter is ignored in mode 1.



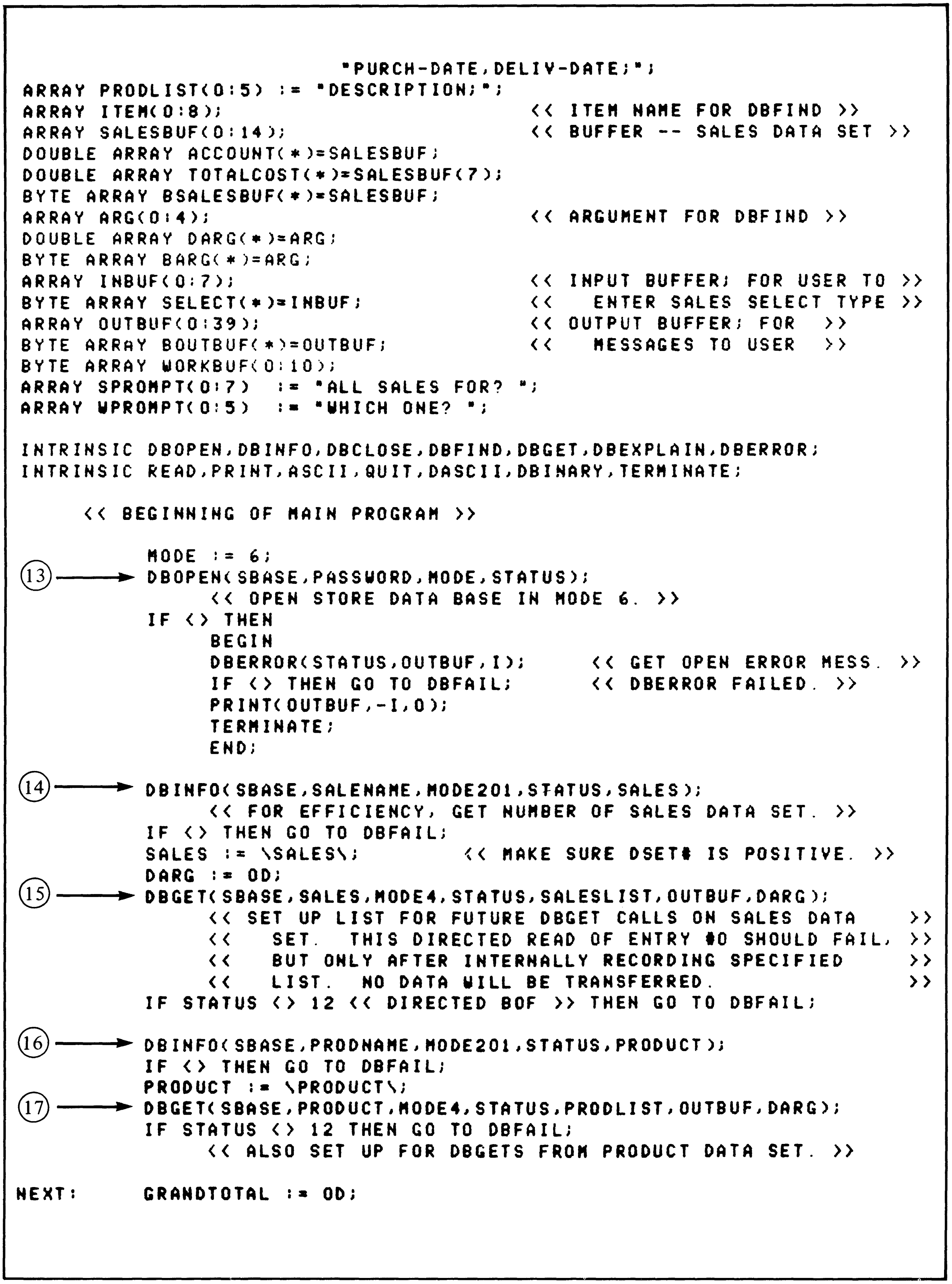
```
:RUN SUPPLMOD


FUNCTION? ADD
SUPPLIER? ACME WIDGET
STREET? 2587 BIRD ST.
CITY? INDIANOLA
STATE? IA
ZIP? 50125

FUNCTION? ADD
SUPPLIER? ACME WIDGET
STREET? 140 CORYDON AVE.
CITY? BROOKLYN
STATE? NY
ZIP? 11208
CAN'T ADD: DUPLICATE SUPPLIER NAME

FUNCTION? CHA
SUPPLIER? ACME WIDGET
STREET? 140 CORYDON AVE.
CITY? BROOKLYN
STATE? NY
ZIP? 11208

FUNCTION? DELETE
SUPPLIER? ACME WIDGET

FUNCTION? DEL
SUPPLIER? ACME WIDGET
NO SUCH SUPPLIER

FUNCTION? /E

 END OF PROGRAM
```

Figure 5-4. Sample SUPPLMOD Execution



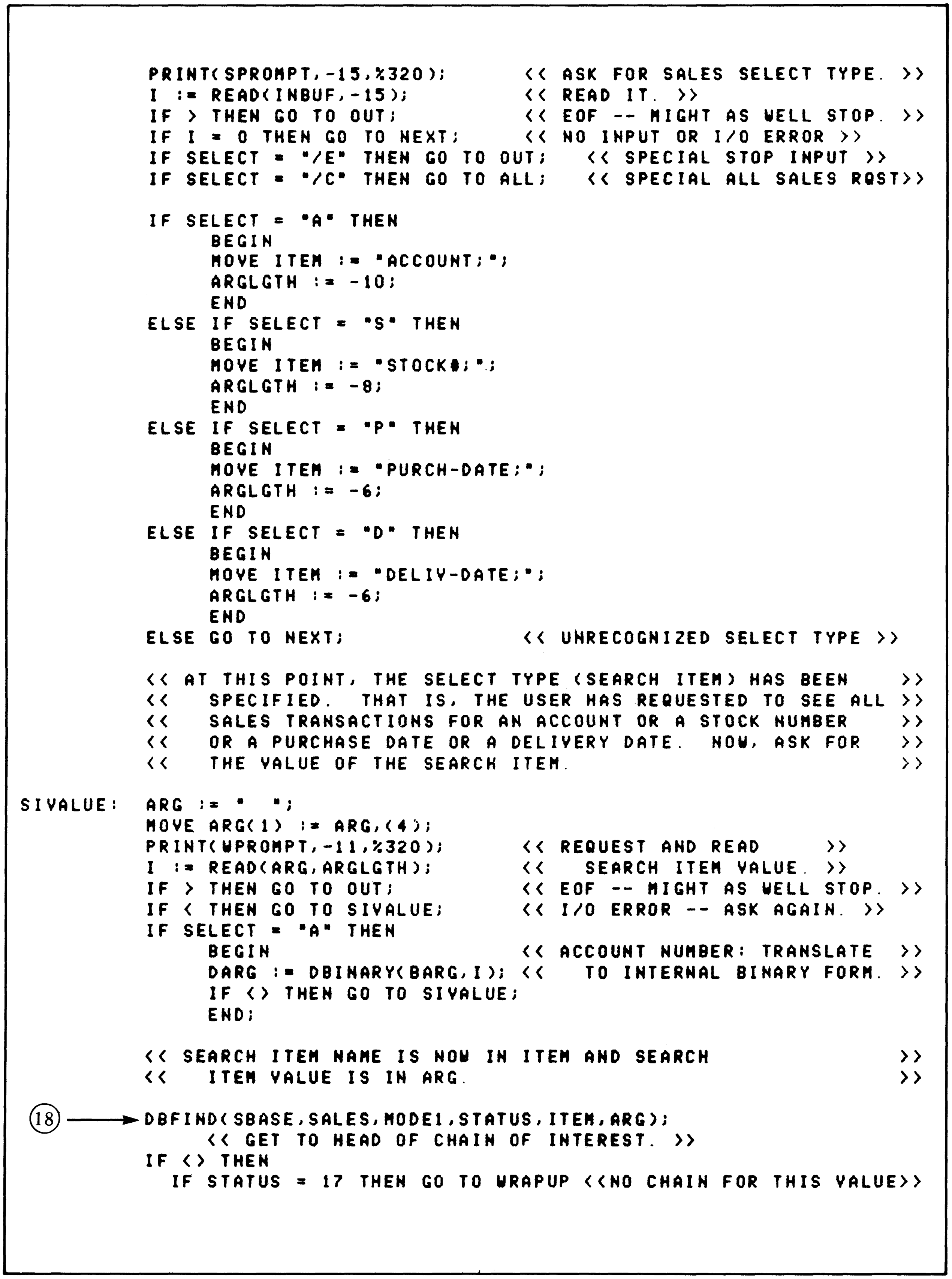
```
BEGIN

<<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
<< THIS PROGRAM OPENS THE "STORE" DATA BASE IN                            >>
<<   MODE 6 AND ALLOWS THE USER INTERACTIVELY TO REQUEST DISPLAYS         >>
<<   OF SALES TRANSACTIONS FROM THE SALES DATA SET.  FOR EACH             >>
<<   TRANSACTION, THE ITEMS FROM WITHIN THE CORRESPONDING ENTRY WHICH     >>
<<   ARE DISPLAYED ARE ACCOUNT, QUANTITY, STOCK#, TOTAL (TOTAL PRICE      >>
<<   IN PENNIES), PURCH-DATE, AND DELIV-DATE.  ALSO, THE DESCRIPTION      >>
<<   OF THE PRODUCT IS OBTAINED FROM THE PRODUCT DATA SET AND PRINTED     >>
<<   NEXT TO THE STOCK#  AFTER THE SALES LINES, A GRAND TOTAL PRICE       >>
<<   LINE IS PRINTED.                                                     >>
<< THERE ARE FIVE WAYS OF SELECTING SALES ENTRIES TO BE PRINTED.          >>
<<   THEY ARE: 1) ALL SALES TRANSACTIONS IN THE DATA SET                  >>
<<             2) ALL SALES TO A PARTICULAR ACCOUNT (CUSTOMER)            >>
<<             3) ALL SALES OF A PARTICULAR STOCK# (PRODUCT)              >>
<<             4) ALL SALES WITH A PARTICULAR PURCHASE DATE               >>
<<             5) ALL SALES WITH A PARTICULAR DELIVERY DATE               >>
<<   FOR (1) ABOVE, THE DATA SET IS READ SERIALLY, WITH EACH ENTRY        >>
<<   ENCOUNTERED BEING PRINTED.  THE OTHER SELECTION METHODS REQUIRE      >>
<<   A SPECIFIC VALUE FOR A CERTAIN ITEM WITHIN THE ENTRY.  SINCE ALL     >>
<<   OF THE ITEMS IN QUESTION ARE SEARCH ITEMS WITHIN THE SALES DATA      >>
<<   SET, THE PROGRAM MERELY CALLS DBFIND FOR THE PARTICULAR ITEM AND     >>
<<   VALUE, AND THEN DOES CHAINED DBGETS TO RETRIEVE THE DESIRED          >>
<<   ENTRIES.  BEFORE DOING SO, OF COURSE THE PROGRAM PROMPTS THE         >>
<<   USER FOR THE ITEM NAME AND ITS VALUE.                                >>
<< THE PROGRAM CAN BE RUN FROM MULTIPLE TERMINALS (SESSIONS)              >>
<<   SIMULTANEOUSLY, AND CAN ACCESS THE DATA BASE CONCURRENTLY WITH       >>
<<   OTHER PROGRAMS WHICH OPEN IT IN MODE 2, 4, 6, OR 8.                  >>
<<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>

INTEGER    MODE,
           MODE1      := 1,     << MODES        >>
           MODE2      := 2,     <<  FOR         >>
           MODE3      := 3,     <<   USE        >>
           MODE4      := 4,     <<    IN        >>
           MODE5      := 5,     <<     IMAGE    >>
           MODE7      := 7,     <<      CALLS   >>
           MODE201    := 201,
           SALES,               << DATA SET NUMBER -- SALES      >>
           PRODUCT,             << DATA SET NUMBER -- PRODUCT    >>
           ARGLGTH,
           I;                   << HANDY-DANDY VARIABLE >>

LOGICAL    SAMELIST   := "*;";  << SPECIAL "SAME AS LAST TIME" LIST >>

DOUBLE GRANDTOTAL;

ARRAY SBASE(0:3)    := "  STORE;";       << DATA BASE >>
ARRAY PASSWORD(0:2)  := "CLERK;";        << QUALIFIER -- PASSWORD >>
ARRAY SALENAME(0:2) := "SALES;";         << QUALIFIER -- DATA SET NAME >>
ARRAY PRODNAME(0:3) := "PRODUCT;";       << QUALIFIER -- DATA SET NAME >>
ARRAY STATUS(0:9);                       << STATUS AREA >>
ARRAY SALESLIST(0:25) := "ACCOUNT,QUANTITY,STOCK#,TOTAL,",
```

Figure 5-5. Purchase Transaction Display Program



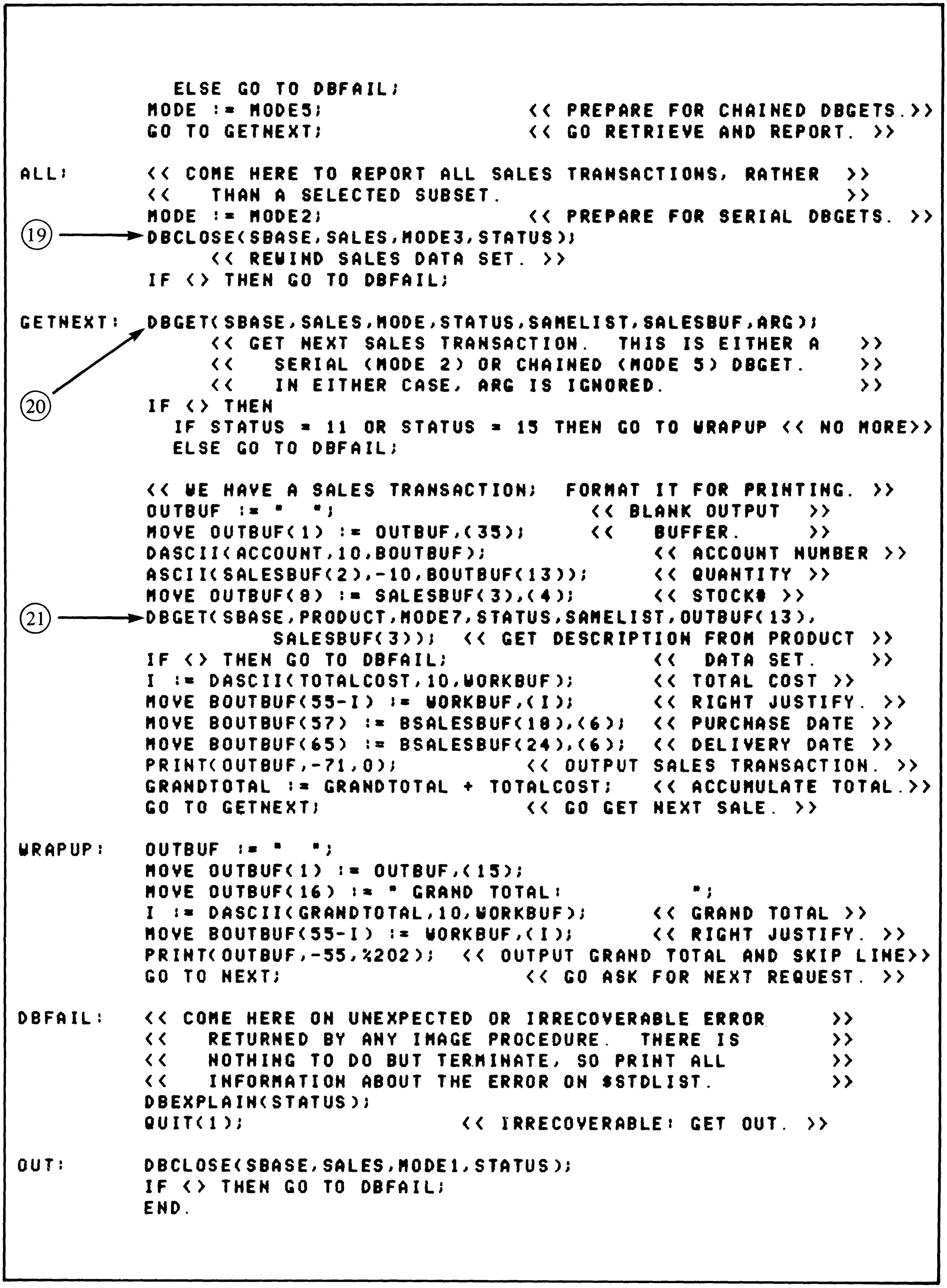
```
                             "PURCH-DATE,DELIV-DATE;";
ARRAY PRODLIST(0:5) := "DESCRIPTION;";
ARRAY ITEM(0:8);                           << ITEM NAME FOR DBFIND >>
ARRAY SALESBUF(0:14);                      << BUFFER -- SALES DATA SET >>
DOUBLE ARRAY ACCOUNT(*)=SALESBUF;
DOUBLE ARRAY TOTALCOST(*)=SALESBUF(7);
BYTE ARRAY BSALESBUF(*)=SALESBUF;
ARRAY ARG(0:4);                            << ARGUMENT FOR DBFIND >>
DOUBLE ARRAY DARG(*)=ARG;
BYTE ARRAY BARG(*)=ARG;
ARRAY INBUF(0:7);                          << INPUT BUFFER; FOR USER TO >>
BYTE ARRAY SELECT(*)=INBUF;                <<   ENTER SALES SELECT TYPE >>
ARRAY OUTBUF(0:39);                        << OUTPUT BUFFER; FOR  >>
BYTE ARRAY BOUTBUF(*)=OUTBUF;              <<   MESSAGES TO USER  >>
BYTE ARRAY WORKBUF(0:10);
ARRAY SPROMPT(0:7)  := "ALL SALES FOR? ";
ARRAY WPROMPT(0:5)  := "WHICH ONE? ";

INTRINSIC DBOPEN,DBINFO,DBCLOSE,DBFIND,DBGET,DBEXPLAIN,DBERROR;
INTRINSIC READ,PRINT,ASCII,QUIT,DASCII,DBINARY,TERMINATE;

    << BEGINNING OF MAIN PROGRAM >>

        MODE := 6;
(13) -> DBOPEN(SBASE,PASSWORD,MODE,STATUS);
            << OPEN STORE DATA BASE IN MODE 6. >>
        IF <> THEN
            BEGIN
            DBERROR(STATUS,OUTBUF,I);      << GET OPEN ERROR MESS. >>
            IF <> THEN GO TO DBFAIL;       << DBERROR FAILED. >>
            PRINT(OUTBUF,-I,0);
            TERMINATE;
            END;

(14) -> DBINFO(SBASE,SALENAME,MODE201,STATUS,SALES);
            << FOR EFFICIENCY, GET NUMBER OF SALES DATA SET. >>
        IF <> THEN GO TO DBFAIL;
        SALES := \SALES\;          << MAKE SURE DSET# IS POSITIVE. >>
        DARG := 0D;
(15) -> DBGET(SBASE,SALES,MODE4,STATUS,SALESLIST,OUTBUF,DARG);
            << SET UP LIST FOR FUTURE DBGET CALLS ON SALES DATA    >>
            <<   SET.  THIS DIRECTED READ OF ENTRY #0 SHOULD FAIL, >>
            <<   BUT ONLY AFTER INTERNALLY RECORDING SPECIFIED     >>
            <<   LIST.  NO DATA WILL BE TRANSFERRED.               >>
        IF STATUS <> 12 << DIRECTED BOF >> THEN GO TO DBFAIL;

(16) -> DBINFO(SBASE,PRODNAME,MODE201,STATUS,PRODUCT);
        IF <> THEN GO TO DBFAIL;
        PRODUCT := \PRODUCT\;
(17) -> DBGET(SBASE,PRODUCT,MODE4,STATUS,PRODLIST,OUTBUF,DARG);
        IF STATUS <> 12 THEN GO TO DBFAIL;
            << ALSO SET UP FOR DBGETS FROM PRODUCT DATA SET. >>

NEXT:   GRANDTOTAL := 0D;
```

Figure 5-5. Purchase Transaction Display Program (Continued)



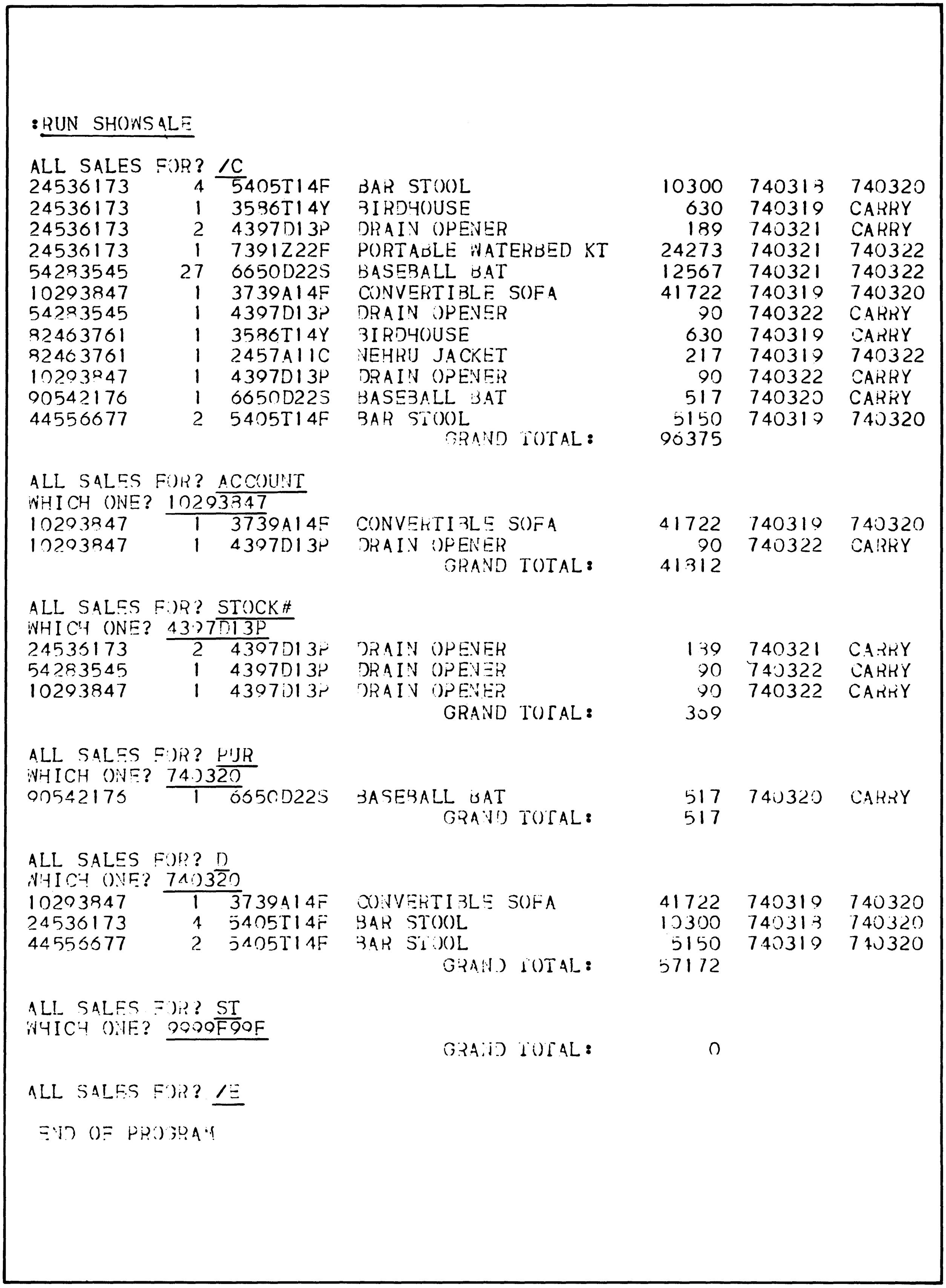
```
          PRINT(SPROMPT,-15,%320);      << ASK FOR SALES SELECT TYPE. >>
          I := READ(INBUF,-15);         << READ IT. >>
          IF > THEN GO TO OUT;          << EOF -- MIGHT AS WELL STOP. >>
          IF I = 0 THEN GO TO NEXT;     << NO INPUT OR I/O ERROR >>
          IF SELECT = "/E" THEN GO TO OUT;   << SPECIAL STOP INPUT >>
          IF SELECT = "/C" THEN GO TO ALL;   << SPECIAL ALL SALES RQST>>

          IF SELECT = "A" THEN
               BEGIN
               MOVE ITEM := "ACCOUNT;";
               ARGLGTH := -10;
               END
          ELSE IF SELECT = "S" THEN
               BEGIN
               MOVE ITEM := "STOCK#;";
               ARGLGTH := -8;
               END
          ELSE IF SELECT = "P" THEN
               BEGIN
               MOVE ITEM := "PURCH-DATE;";
               ARGLGTH := -6;
               END
          ELSE IF SELECT = "D" THEN
               BEGIN
               MOVE ITEM := "DELIV-DATE;";
               ARGLGTH := -6;
               END
          ELSE GO TO NEXT;              << UNRECOGNIZED SELECT TYPE >>

          << AT THIS POINT, THE SELECT TYPE (SEARCH ITEM) HAS BEEN     >>
          <<   SPECIFIED.  THAT IS, THE USER HAS REQUESTED TO SEE ALL >>
          <<   SALES TRANSACTIONS FOR AN ACCOUNT OR A STOCK NUMBER     >>
          <<   OR A PURCHASE DATE OR A DELIVERY DATE.  NOW, ASK FOR    >>
          <<   THE VALUE OF THE SEARCH ITEM.                           >>

SIVALUE:  ARG := "  ";
          MOVE ARG(1) := ARG,(4);
          PRINT(WPROMPT,-11,%320);      << REQUEST AND READ      >>
          I := READ(ARG,ARGLGTH);       <<   SEARCH ITEM VALUE. >>
          IF > THEN GO TO OUT;          << EOF -- MIGHT AS WELL STOP. >>
          IF < THEN GO TO SIVALUE;      << I/O ERROR -- ASK AGAIN. >>
          IF SELECT = "A" THEN
               BEGIN                    << ACCOUNT NUMBER: TRANSLATE  >>
               DARG := DBINARY(BARG,I); <<   TO INTERNAL BINARY FORM. >>
               IF <> THEN GO TO SIVALUE;
               END;

          << SEARCH ITEM NAME IS NOW IN ITEM AND SEARCH              >>
          <<   ITEM VALUE IS IN ARG.                                 >>

  ⑱ ——► DBFIND(SBASE,SALES,MODE1,STATUS,ITEM,ARG);
               << GET TO HEAD OF CHAIN OF INTEREST. >>
          IF <> THEN
            IF STATUS = 17 THEN GO TO WRAPUP <<NO CHAIN FOR THIS VALUE>>
```

Figure 5-5. Purchase Transaction Display Program (Continued)



```
             ELSE GO TO DBFAIL;
           MODE := MODE5;               << PREPARE FOR CHAINED DBGETS.>>
           GO TO GETNEXT;               << GO RETRIEVE AND REPORT. >>

ALL:       << COME HERE TO REPORT ALL SALES TRANSACTIONS, RATHER  >>
           <<   THAN A SELECTED SUBSET.                           >>
           MODE := MODE2;               << PREPARE FOR SERIAL DBGETS. >>
(19)       DBCLOSE(SBASE,SALES,MODE3,STATUS);
                << REWIND SALES DATA SET. >>
           IF <> THEN GO TO DBFAIL;

GETNEXT:   DBGET(SBASE,SALES,MODE,STATUS,SAMELIST,SALESBUF,ARG);
                << GET NEXT SALES TRANSACTION.  THIS IS EITHER A    >>
                <<    SERIAL (MODE 2) OR CHAINED (MODE 5) DBGET.    >>
                <<    IN EITHER CASE, ARG IS IGNORED.               >>
(20)       IF <> THEN
             IF STATUS = 11 OR STATUS = 15 THEN GO TO WRAPUP << NO MORE>>
             ELSE GO TO DBFAIL;

           << WE HAVE A SALES TRANSACTION;  FORMAT IT FOR PRINTING. >>
           OUTBUF := "  ";                 << BLANK OUTPUT   >>
           MOVE OUTBUF(1) := OUTBUF,(35);  <<   BUFFER.      >>
           DASCII(ACCOUNT,10,BOUTBUF);          << ACCOUNT NUMBER >>
           ASCII(SALESBUF(2),-10,BOUTBUF(13));  << QUANTITY >>
           MOVE OUTBUF(8) := SALESBUF(3),(4);   << STOCK# >>
(21)       DBGET(SBASE,PRODUCT,MODE7,STATUS,SAMELIST,OUTBUF(13),
                     SALESBUF(3));  << GET DESCRIPTION FROM PRODUCT >>
           IF <> THEN GO TO DBFAIL;             <<  DATA SET.     >>
           I := DASCII(TOTALCOST,10,WORKBUF);   << TOTAL COST >>
           MOVE BOUTBUF(55-I) := WORKBUF,(I);   << RIGHT JUSTIFY. >>
           MOVE BOUTBUF(57) := BSALESBUF(18),(6); << PURCHASE DATE >>
           MOVE BOUTBUF(65) := BSALESBUF(24),(6); << DELIVERY DATE >>
           PRINT(OUTBUF,-71,0);         << OUTPUT SALES TRANSACTION. >>
           GRANDTOTAL := GRANDTOTAL + TOTALCOST;  << ACCUMULATE TOTAL.>>
           GO TO GETNEXT;               << GO GET NEXT SALE. >>

WRAPUP:    OUTBUF := "  ";
           MOVE OUTBUF(1) := OUTBUF,(15);
           MOVE OUTBUF(16) := " GRAND TOTAL:          ";
           I := DASCII(GRANDTOTAL,10,WORKBUF);  << GRAND TOTAL >>
           MOVE BOUTBUF(55-I) := WORKBUF,(I);   << RIGHT JUSTIFY. >>
           PRINT(OUTBUF,-55,%202);  << OUTPUT GRAND TOTAL AND SKIP LINE>>
           GO TO NEXT;                  << GO ASK FOR NEXT REQUEST. >>

DBFAIL:    << COME HERE ON UNEXPECTED OR IRRECOVERABLE ERROR     >>
           <<   RETURNED BY ANY IMAGE PROCEDURE.  THERE IS       >>
           <<   NOTHING TO DO BUT TERMINATE, SO PRINT ALL        >>
           <<   INFORMATION ABOUT THE ERROR ON $STDLIST.         >>
           DBEXPLAIN(STATUS);
           QUIT(1);                 << IRRECOVERABLE! GET OUT. >>

OUT:       DBCLOSE(SBASE,SALES,MODE1,STATUS);
           IF <> THEN GO TO DBFAIL;
           END.
```

Figure 5-5. Purchase Transaction Display Program (Continued)



```
:RUN SHOWSALE

ALL SALES FOR? /C
24536173     4  5405T14F   BAR STOOL                10300   740318   740320
24536173     1  3586T14Y   BIRDHOUSE                  630   740319   CARRY
24536173     2  4397D13P   DRAIN OPENER               189   740321   CARRY
24536173     1  7391Z22F   PORTABLE WATERBED KT     24273   740321   740322
54283545    27  6650D22S   BASEBALL BAT             12567   740321   740322
10293847     1  3739A14F   CONVERTIBLE SOFA         41722   740319   740320
54283545     1  4397D13P   DRAIN OPENER                90   740322   CARRY
82463761     1  3586T14Y   BIRDHOUSE                  630   740319   CARRY
82463761     1  2457A11C   NEHRU JACKET               217   740319   740322
10293847     1  4397D13P   DRAIN OPENER                90   740322   CARRY
90542176     1  6650D22S   BASEBALL BAT               517   740320   CARRY
44556677     2  5405T14F   BAR STOOL                 5150   740319   740320
                                  GRAND TOTAL:      96375

ALL SALES FOR? ACCOUNT
WHICH ONE? 10293847
10293847     1  3739A14F   CONVERTIBLE SOFA         41722   740319   740320
10293847     1  4397D13P   DRAIN OPENER                90   740322   CARRY
                                  GRAND TOTAL:      41812

ALL SALES FOR? STOCK#
WHICH ONE? 4397D13P
24536173     2  4397D13P   DRAIN OPENER               189   740321   CARRY
54283545     1  4397D13P   DRAIN OPENER                90   740322   CARRY
10293847     1  4397D13P   DRAIN OPENER                90   740322   CARRY
                                  GRAND TOTAL:        369

ALL SALES FOR? PUR
WHICH ONE? 740320
90542176     1  6650D22S   BASEBALL BAT               517   740320   CARRY
                                  GRAND TOTAL:        517

ALL SALES FOR? D
WHICH ONE? 740320
10293847     1  3739A14F   CONVERTIBLE SOFA         41722   740319   740320
24536173     4  5405T14F   BAR STOOL                10300   740318   740320
44556677     2  5405T14F   BAR STOOL                 5150   740319   740320
                                  GRAND TOTAL:      57172

ALL SALES FOR? ST
WHICH ONE? 9999F99F
                                  GRAND TOTAL:          0

ALL SALES FOR? /E

 END OF PROGRAM
```

Figure 5-6. Sample SHOWSALE Execution

⑬ OPEN DATA BASE

The STORE data base is opened in mode 6 with password CLERK.

⑭ REQUEST DATA SET INFORMATION

The data set number for SALES is requested. Note that SALENAME is an array containing "SALES;" and the data set number is stored in the SALES variable.

⑮ READ ENTRY (DIRECTLY)

This call requests a directed read of the entry in record 0. Although this read fails and returns condition word 12, the internal list of items is set up to include ACCOUNT, QUANTITY, STOCK#, and TOTAL. No data is transferred. Subsequent calls to read an entry from the SALES data set can use the special list construct *; indicating the list is the same as the one used in this call. This technique saves processing time since the list is set up only once during program execution.

⑯ REQUEST DATA SET INFORMATION

The data set number for PRODUCT is requested.

⑰ READ ENTRY (DIRECTLY)

This call is the same as ⑮ except the data set name is PRODUCT and the list includes only the DESCRIPTION item.

⑱ READ ENTRY (CHAINED)

A call to DBFIND locates the pointers for a chain in the SALES data set. In the preceding code, the user is prompted for the search item (ACCOUNT, STOCK#, PURCH-DATE, or DELIV-DATE) and its value. ITEM contains the search item name and ARG contains the search item value. If the condition word is 17, there is no chain with the requested value. The read is performed with the call described below in ⑳ .

⑲ REWIND DATA SET

A call to DBCLOSE with mode 3 rewinds the SALES data set to prepare for serial reads of all entries in the set. If the rewind fails, the condition code is CCL or CCG.

⑳ READ ENTRY (SERIAL OR CHAINED)

This call is coded so that it performs either a forward chained (mode 5) or forward serial (mode 2) read of the SALES data set. The data is read into SALESBUF and the list is *; indicating it is the same list that was set up in calls ⑮ and ⑰ . ARG is ignored in both modes 2 and 5. If the end of the data set is reached while doing a serial read, the condition word is 11. If the end of chain is reached while doing a chained read, the condition word is 15. Since access mode 6 and password CLERK allow the user to read all items in the SALES and PRODUCT data sets, it is not necessary to check for condition word –21.

㉑ READ ENTRY (CALCULATED)

A calculated read (mode 7) is performed using the search item value for STOCK# that is in SALESBUF(3) and (4). The data set is PRODUCT and the list parameter is *;. The description is read into OUTBUF(13) through OUTBUF(22).

BASIC EXAMPLES

To simplify your access to an IMAGE data base through BASIC language programs, it is recommended that you use the BIMAGE interface procedures provided with the IMAGE software. These routines convert all parameter byte addresses to word addresses as required by IMAGE. In addition to calling the necessary IMAGE procedure, the BIMAGE procedures perform the following functions for your convenience:

- automatically pack into a buffer a list of expressions before calling the DBPUT or DBUPDATE procedures
- automatically unpack from a buffer to a list of BASIC variables the values of items returned by DBGET or the values returned by DBINFO
- automatically update the logical length of string variables to which data is transferred from the data base to reflect the length of the string actually transferred.

Table 5-1 lists the BIMAGE interface procedures with the IMAGE procedures to which they correspond. The parameters are described in table 5-2. The corresponding IMAGE procedure parameter is listed next to the BIMAGE parameter.

Table 5-1. BIMAGE Procedure Calls

BIMAGE	CORRESPONDS TO:
XDBOPEN (B\$, W\$, mode, status(*))	DBOPEN
XDBPUT (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*), $\left\{\begin{matrix} L\$ \\ l \end{matrix}\right\}$, writelist)	DBPUT
XDBFIND (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*), $\left\{\begin{matrix} I\$ \\ i \end{matrix}\right\}$, $\left\{\begin{matrix} A\$ \\ a \end{matrix}\right\}$)	DBFIND
XDBGET (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*), $\left\{\begin{matrix} L\$ \\ l \end{matrix}\right\}$, readlist, $\left\{\begin{matrix} A\$ \\ a \end{matrix}\right\}$)	DBGET
XDBUPDATE (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*), $\left\{\begin{matrix} L\$ \\ l \end{matrix}\right\}$, writelist)	DBUPDATE
XDBDELETE (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*))	DBDELETE
XDBLOCK (B\$, $\left\{\text{descriplist}\right\}$, mode, status(*))	DBLOCK
XDBUNLOCK (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*))	DBUNLOCK
XDBCLOSE (B\$, $\left\{\begin{matrix} D\$ \\ d \end{matrix}\right\}$, mode, status(*))	DBCLOSE
XDBINFO (B\$, $\left\{\begin{matrix} Q\$ \\ q \end{matrix}\right\}$, mode, status(*), readlist)	DBINFO
XDBEXPLAIN (status(*))	DBEXPLAIN
XDBERROR (status(*), M\$ [,length])	DBERROR

Table 5-2. BIMAGE Procedure Parameters

BIMAGE**	IMAGE	
A$	*argument*	May be any string expression.
a	*argument*	May be a numeric expression or numeric array of any data-type.
B$	*base*	Must be a simple string variable. Value should not be altered between calls to XDBOPEN and XDBCLOSE.
D$	*dset*	May be any string expression.
d	*dset*	May be a type-INTEGER expression.*
descriplist	*qualifier*	Has same form as *writelist*. You should ensure that once BASIC has concatenated the component variables, the result is a valid lock descriptor list (or set name) as defined for DBLOCK. (Parameter ignored for DBLOCK modes 1 and 2)
I$	*item*	May be any string expression.
i	*item*	May be a type-INTEGER expression.*
L$	*list*	May be any string expression or a string array. If it is a string array, all of the string elements are concatenated to form one string whose length may not exceed 255 characters. The concatenated string must form a syntactically correct *list* parameter. Commas must be placed appropriately.
l	*list*	May be an array of type INTEGER.
length	*length*	Must be a simple or subscripted type-INTEGER variable (if not, parameter is ignored.) Parameter is optional but if present, total length of IMAGE message is returned. Value may exceed length of message by BIMAGE procedure if *M$* is too small and message is truncated. Not needed when M$ is a string variable.
M$	*buffer*	Should be a simple or subscripted string variable without substring designators. If message is larger than *M$*, message is truncated on the right. Logical length of *M$* is set to length of message returned by BIMAGE and may not be equal to *length* if message is truncated.
mode	*mode*	Must be type-INTEGER expression.*
Q$	*qualifier*	May be any string expression.
q	*qualifier*	May be a type-INTEGER expression.*
status	*status*	Must be a type-INTEGER array containing at least ten active elements.
W$	*password*	May be any string expression.
readlist	*buffer*	Has form similar to *item list* of BASIC READ or MAT READ statement. May consist of one or more string or numeric simple or subscripted variables or arrays separated by commas. String variables with substring designators and the "FOR-loop" construct are not permitted.
writelist	*buffer*	Has form similar to *item list* of BASIC PRINT or MAT PRINT statement. May consist of one or more string or numeric expressions or arrays separated by commas. "FOR-loop" not permitted. Substring designators are permitted.

* See discussion of type-INTEGER expressions as parameters.

** Note that if you specify an array as a parameter you must obey BASIC syntax rules and append parentheses and asterisks, for example, L$(*,*) or A(*).

BASIC

Refer to the IMAGE procedure descriptions in Section IV for details regarding the purpose of a procedure and its parameters as well as available options.

BIMAGE provides some extensions to the IMAGE procedure calling sequences to simplify your access to the data base:

- BIMAGE allows you to enter a list of expressions in place of the *buffer* parameter. The list is automatically packed into or unpacked from a temporary buffer constructed by the BIMAGE procedures. This facility is also available to construct lock descriptor lists.

- String or numeric expressions are accepted for many parameters. For example, the *dset* parameter may be a string expression when specifying the data set by name or a numeric expression when specifying the data set by number.

STRING VARIABLES

The *physical* length of a string variable determines the number of characters (bytes) read by the XDBGET procedure and the *logical* length of a string variable determines the number of characters written by the XDBPUT and XDBUPDATE procedures. Thus, you should ensure that the physical length of a string variable specified in a DIM or COM statement exactly matches the size of the item to be read by a call to XDBGET.

On the other hand, the same string variable can be used to write items of varying sizes. Substring designators should be used to ensure that the actual string passed to XDBPUT or XDBUPDATE fills the item to be written. For example, if the item is 8 characters long, and substring S$(3) is 2 characters long, S$(3,10) or S$(3;8) fills the item with the S$(3) substring and appends 6 blanks.

If the string variable is an array, the length of each string element or of the concatenated string elements should correspond to the length of the item or sub-item to be written. You can ensure this by specifying substring designators when assigning values to elements of the string array in your BASIC program.

TYPE-INTEGER EXPRESSIONS AS PARAMETERS

Since BASIC treats integral numeric constants as type-REAL, expressions involving constants cannot be passed directly to a type-INTEGER parameter of a BIMAGE procedure. You can define a function such as the following to ensure that a type-INTEGER expression is passed:

```
10 DEF INTEGER FNI(X)=X
```

When a procedure call is made, the function is used in this way:

```
50 CALL XDBLOCK(B$, D$, FNI(expression),S(*) )
```

The function FNI converts *expression* to type-INTEGER.

THE READLIST, WRITELIST, AND DESCRIPLIST PARAMETERS

When specifying string expressions in a *readlist*, *writelist*, or *descriplist*, each string expression should correspond to a data item or sub-item, or groups of items or sub-items in the case of string arrays. You should not specify several string expressions as the source or destination of one item or sub-item. The transfer of strings to or from the data base always begins on a word boundary of the buffer. Therefore, writing from or reading into two odd-length strings is not the same as writing or reading into one even-length string.

THE STATUS PARAMETER

If the *status* parameter is a type-INTEGER variable, a condition word is returned in the first word and the second word is set to zero if *status* is at least a two-element array. The condition word will have a value equal to those listed for the corresponding IMAGE procedure and, in addition, may contain one of the conditions listed in table 5-3.

If the *status* parameter is not type-INTEGER, the BIMAGE procedures cannot return a condition word for the common error: failure to declare the *status* variable type-INTEGER. This error will usually result in the BASIC message UNDEFINED VALUE the first time the *status* array contents are examined.

Table 5-3. Additional BIMAGE Condition Word Values

EXCEPTIONAL CONDITIONS:		PROCEDURES:
51	Insufficient stack for temporary buffer.	XDBGET,XDBPUT, XDBINFO,XDBUPDATE
52	Invalid number of parameters.	All procedures except XDBERROR and XDBEXPLAIN
53	Invalid parameter.	
54	*status* array has less than 10 elements.	

OPEN DATA BASE

```
  10 DIM B$[8],P$[8]
  20 INTEGER S[10],M
  30 B$="  STORE;"
  40 INPUT "ENTER PASSWORD: ",P$[1;8]
  50 INPUT "ENTER ACCESS MODE (1-8): ",M
  60 CALL XDBOPEN(B$,P$,M,S[*])
  70 IF S[1]<>0 THEN 9300

                 •
                 •
                 •

       (code to use data base)

                 •

9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBEXPLAIN(S[*])
9320 STOP

                 •
                 •
```

In this example, the STORE data base is opened in the access mode entered by the user and with a user class number corresponding to the password entered by the user and stored in the P$ string. If the password is less than 8 characters the P$ string is padded with blanks. The first word of the status array, S, is tested to determine whether the procedure executed successfully. If not, an error message is printed.

BASIC

ADD ENTRY

```
  10 DIM B$[8],P$[8],A$[8],C$[20]
  20 INTEGER S[10],M,M1
  30 B$="  STORE;"
  40 INPUT "ENTER PASSWORD: ",P$[1;8]
  50 INPUT "ENTER ACCESS MODE (3,4): ",M
  60 GOTO M OF 70,70,90,90
  70 PRINT "CANNOT ADD ENTRIES IN THIS ACCESS MODE"
  80 GOTO 50
  90 CALL XDBOPEN(B$,P$,M,S[*])
 100 IF S[1]<>0 THEN 9300
 110 INPUT "ENTER STOCK# OR / TO TERMINATE: ",A$[1;8]
 120 IF A$[1,1]="/" THEN GOTO 9900
 130 INPUT "ENTER DESCRIPTION: ",C$[1;20]
 140 M1=1
 150 CALL XDBPUT(B$,"PRODUCT;",M1,S[*],"@;",A$[1;8],C$[1;20])
 160 IF S[1]<>43 THEN 190
 170 PRINT "DUPLICATE STOCK NUMBER"
 180 GOTO 110
 190 IF S[1]<>16 THEN 220
 200 PRINT "DATA SET FULL"
 210 GOTO 9900
 220 IF S[1]<>0 THEN 250
 230 PRINT "NEW PRODUCT HAS BEEN ENTERED"
 240 GOTO 110
 250 IF S[1]=-23 THEN 290
 260 PRINT "DBPUT FAILURE"
 270 CALL XDBEXPLAIN(S[*])
 280 GOTO 9900
 290 PRINT "YOUR PASSWORD DOES NOT ALLOW YOU TO ADD ENTRIES"
 300 GOTO 9900

             •
             •
```

After data base opened, first word of B$ contains data segment number.

9300 (code same as example above)

9900 (close data base)

This sample code adds an entry to the PRODUCT manual master data set. Note that the B$ string used to open the data base is the *base* parameter in this call. It should not be changed after the call to XDBOPEN since this call saves a data segment number in the first word of B$. The list of items to be added is specified as @; which indicates that values are specified for all items in the entry. The values for the STOCK# and DESCRIPTION data items are stored in A$ and C$. Sample values are "7474Z74Z" and "ORANGE CRATEΔΔΔΔΔΔΔΔ".

In the example, the condition word of the status array is tested for a value of 43, indicating that an entry with the specified STOCK# search item value already exists in the data set, or 16, indicating that the data set is full, or –23, indicating that the user's password does not grant write access to the data set.

If an entry is to be added to a detail set, the program may first check to see if the required entries exist in the manual masters linked to the detail set. Values must be provided for all search items and the sort item, if one is defined, of a detail data set entry.

READ ENTRY (SERIALLY)

```
 10 DIM B$[8],P$[8],D1$[14],L1$[20],S1$[16],S2$[2]
 20 INTEGER S[10],M,M1,M2
 30 B$="  STORE;"
 40 M1=1
 50 INPUT "ENTEP PASSWORD: ",P$[1;8]
 60 INPUT "ENTER ACCESS MODE (1-8): ",M
 70 CALL XDBOPEN(B$,P$,M,S[*])
 80 IF S[1]<>0 THEN 9300
200 M2=2
210 D1$="SUP-MASTER;"
220 L1$="SUPPLIER,STATE;"◄─────────────────────readlist
230 CALL XDBGET(B$,D1$,M2,S[*],L1$,S1$,S2$,"")
240 IF S[1]<>11 THEN 270
250 GOSUB 900
260 GOTO 230
270 IF S[1]<>0 THEN 320
280 PRINT "SUPPLIER= ",S1$,"STATE= ",S2$
290 INPUT "CONTINUE (Y OP N)? ",X$
300 IF X$[1,1]="Y" THEN GOTO 230
310 GOTO 9900
320 IF S[1]=-21 THEN 360
330 PRINT "DBGET FAILURE"
340 CALL XDBEXPLAIN(S[*])
350 GOTO 9900
360 PRINT "YOU DO NOT HAVE ACCESS TO THIS DATA"
370 GOTO 9900
          •
900 (routine to rewind data set)


9300 (same as XDBOPEN example)
          •
          •
9900 (close data base)
```

To read the next entry of the SUP-MASTER data set, a mode of 2 is used. This directs the XDBGET (and DBGET) procedure to perform a forward serial read. In the example, the list in the L1$ string specifies two data items to be read. After returning to the calling program, the S1$ string contains the STOCK# data item value and S2$ contains the DESCRIPTION data item value. The *argument* parameter is ignored if *mode* equals 2, therefore, a null string may be used for this parameter.

If an end-of-file is encountered the condition word is set to 11. In this case, if the user wants to continue, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the XDBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read will read the first entry in the data set.

If the user's password does not allow read access to the data, a condition word of –21 is returned.

READ ENTRY (CALCULATED)

```
 10 DIM B$[8],P$[8],C$[20],S0$[8]
 20 INTEGER S[10],M1,M
 30 B$="  STORE;"
 40 M1=1
 50 DEF INTEGER FNI(X)=X
 60 INPUT "ENTER PASSWORD: ",P$[1;8]
 70 INPUT "ENTER ACCESS MODE (1-8): ",M
 80 CALL XDBOPEN(B$,P$,M,S[*])
 90 IF S[1]<>0 THEN 9300
300 INPUT "ENTER STOCK# OR / TO TERMINATE: ",S0$[1;8]
310 IF S0$[1,1]="/" THEN GOTO 9900
320 CALL XDBGET(B$,"PRODUCT ",FNI(7),S[*],"DESCRIPTION;",C$,S0$)
330 IF S[1]<>17 THEN GOTO 360
340 PRINT "NO SUCH STOCK NUMBER"
350 GOTO 300
360 IF S[1]=0 THEN GOTO 410
370 IF S[1]=-21 THEN 430
380 PRINT "DBGET FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 PRINT S0$,C$
420 GOTO 300
430 PRINT "YOUR PASSWORD DOES NOT GRANT ACCESS TO DATA REQUESTED"
440 GOTO 9900
        •
        •
```

9300 (same code as XDBOPEN example)

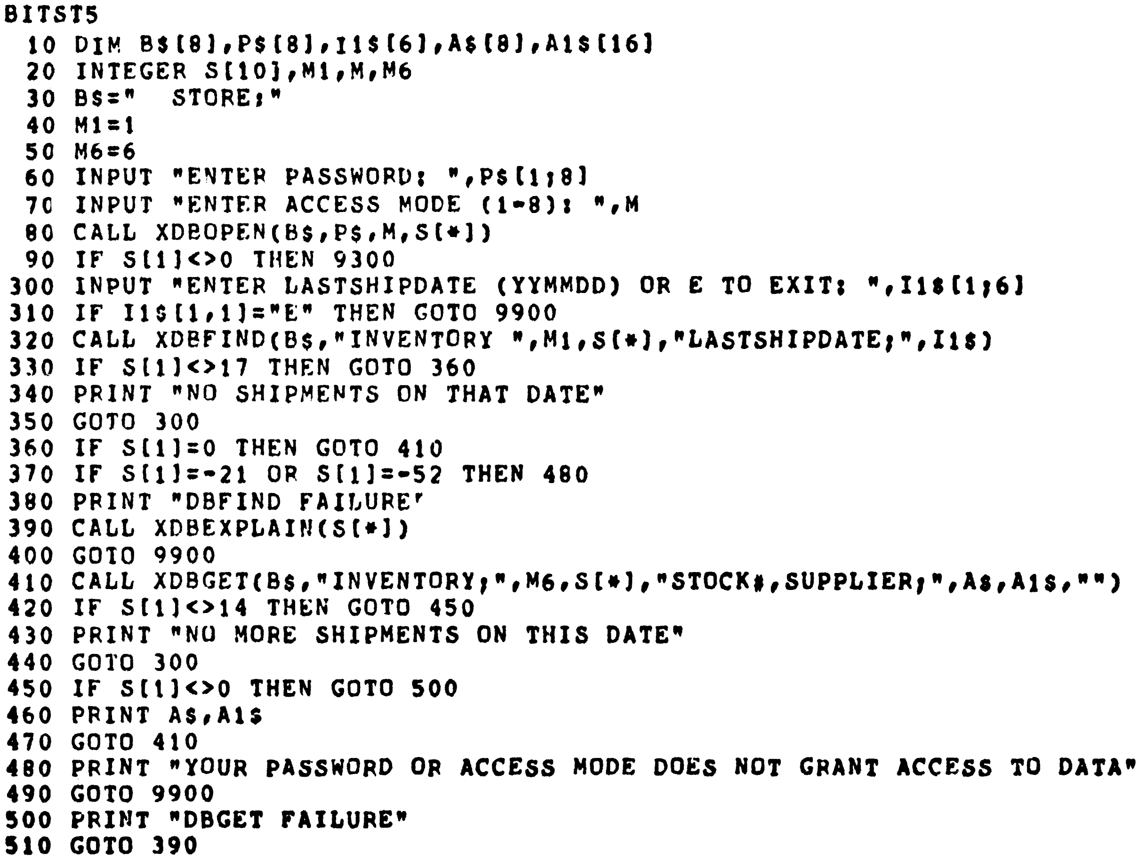
        •

9900 (close data base)

        •
        •

To locate the PRODUCT data set entry which has a STOCK# search item value equal to the one entered in S0$ by user, a calculated read is used. The mode is 7 and the item to be read is DESCRIPTION. After XDBGET returns control to the calling program, the description is in C$. If no entry exists with the specified STOCK# value, the condition word is 17. If the user does not have read access to the requested data, a condition word of –21 is returned.

READ ENTRY (BACKWARD CHAIN)



```
BITST5
 10 DIM B$[8],P$[8],I1$[6],A$[8],A1$[16]
 20 INTEGER S[10],M1,M,M6
 30 B$="  STORE;"
 40 M1=1
 50 M6=6
 60 INPUT "ENTER PASSWORD: ",P$[1;8]
 70 INPUT "ENTER ACCESS MODE (1-8): ",M
 80 CALL XDBOPEN(B$,P$,M,S[*])
 90 IF S[1]<>0 THEN 9300
300 INPUT "ENTER LASTSHIPDATE (YYMMDD) OR E TO EXIT: ",I1$[1;6]
310 IF I1$[1,1]="E" THEN GOTO 9900
320 CALL XDBFIND(B$,"INVENTORY ",M1,S[*],"LASTSHIPDATE;",I1$)
330 IF S[1]<>17 THEN GOTO 360
340 PRINT "NO SHIPMENTS ON THAT DATE"
350 GOTO 300
360 IF S[1]=0 THEN GOTO 410
370 IF S[1]=-21 OR S[1]=-52 THEN 480
380 PRINT "DBFIND FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 CALL XDBGET(B$,"INVENTORY;",M6,S[*],"STOCK#,SUPPLIER;",A$,A1$,"")
420 IF S[1]<>14 THEN GOTO 450
430 PRINT "NO MORE SHIPMENTS ON THIS DATE"
440 GOTO 300
450 IF S[1]<>0 THEN GOTO 500
460 PRINT A$,A1$
470 GOTO 410
480 PRINT "YOUR PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS TO DATA"
490 GOTO 9900
500 PRINT "DBGET FAILURE"
510 GOTO 390
```

⋮

9300 (same as XDBOPEN example)

⋮

9900 (close data base)

First the XDBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value of both the master entry containing the chain head and the detail entries making up the chain. The search item value is requested from the user and stored in I1$, for example, the user may enter 760314.

If no entry in the DATE-MASTER has a search item value entered, the condition word will be 17. If the user does not have read access to the data, a condition word of –21 or –52 is returned.

If the XDBFIND procedure executes successfully, a call to the XDBGET procedure with a mode parameter of 6 reads the last entry in the chain. Subsequent calls to XDBGET with the same mode read backward through the chain until the first entry has been read. If the condition word is 14, the beginning of the chain has been reached and no more entries are available, or there are no entries in the chain.

If an entry is successfully read, the program uses the STOCK# value stored in A$ and the SUPPLIER value stored in A1$ and then returns to statement 350 to read another entry in the chain.

UPDATE ENTRY

```
 10 DIM B$[8],P$[8],D1$[12],I2$[16],A5$[26],S9$[16]
 20 INTEGER S[10],M
 30 B$="  STORE;"
 40 DEF INTEGER FNI(X)=X
 50 D1$="SUP-MASTER "
 60 I2$="STREET-ADDRESS;"
 70 INPUT "ENTER PASSWORD: ",P$[1;8]
 80 M=3
 90 CALL XDBOPEN(B$,P$,M,S[*])
100 IF S[1]<>0 THEN 9300
200 INPUT "ENTER SUPPLIER OR / TO TERMINATE: ",S9$[1;16]
210 IF S9$[1,1]="/" THEN GOTO 9900
220 CALL XDBGET(B$,D1$,FNI(7),S[*],I2$,A5$,S9$)
230 IF S[1]=-21 THEN GOTO 290
240 IF S[1]=0 THEN GOTO 310
250 IF S[1]=17 THEN GOTO 430
260 PRINT "DBGET FAILURE"
270 CALL XDBEXPLAIN(S[*])
280 GOTO 9900
290 PRINT &
    "YOUR PASSWORD OR ACCESS MODE DOES NOT ALLOW ACCESS TO THIS DATA"
300 GOTO 9900
310 PRINT "CURRENT ADDRESS: ",A5$
320 INPUT "ENTER NEW ADDRESS: ",A5$[1;26]
330 CALL XDBUPDATE(B$,D1$,FNI(1),S[*],I2$,A5$)
340 IF S[1]<>42 THEN GOTO 370
350 PRINT "YOU ARE NOT ALLOWED TO ALTER THIS ITEM"
360 GOTO 200
370 IF S[1]=0 THEN 410
380 PRINT "DBUPDATE FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 PRINT "ADDRESS CHANGED"
420 GOTO 200
430 PRINT "NO SUCH SUPPLIER"
440 GOTO 200
              •
```

9310 (same as XDBOPEN example)

:

9900 (close data base)

Before an entry can be updated it must be located. In this example, the entry is located with a calculated XDBGET that reads the STREET-ADDRESS item in the SUP-MASTER data set. The entry is located by using the SUPPLIER search item with a value supplied by the user. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into A5$. The XDBUPDATE procedure is then called to alter the STREET-ADDRESS data item in the entry.

If the current user class number does not allow this item to be altered or the access mode does not allow updates to take place, the condition word 42 is returned.

A null list can be used with DBGET to locate an entry to be updated.

DELETE ENTRY (WITH LOCKING AND UNLOCKING)

```
 10 DIM B$[8],P$[8],D1$[12],S9$[16],A5$[16]
 20 INTEGER S[10],M2,M1,M4
 30 B$="  STORE;"
 40 DEF INTEGER FNI(X)=X
 50 D1$="SUP-MASTER "
 60 INPUT "ENTER PASSWORD: ",P$[1;8]
 70 M1=1
 80 M2=2
 85 M4=4
 90 CALL XDBOPEN(B$,P$,M1,S[*])
100 IF S[1]<>0 THEN 9300
110 INPUT "ENTER SUPPLIER OR / TO TERMINATE: ",S9$[1;16]
120 IF S9$[1,1]="/" THEN GOTO 9900
130 CALL XDBLOCK(B$,D1$,M4,S[*])
140 IF S[1]<=0 THEN 170
150 PRINT "DATA SET IS BUSY. TRY AGAIN LATER."
160 GOTO 9900
170 IF S[1]=0 THEN 210
180 PRINT "DBLOCK FAILURE"
190 CALL XDBEXPLAIN(S[*])
200 GOTO 9900
210 CALL XDBGET(B$,D1$,FNI(7),S[*],"SUPPLIER;",A5$,S9$)
220 IF S[1]=0 THEN 330
230 IF S[1]=-21 THEN 280
240 IF S[1]=17 THEN 310
250 PRINT "DBGET FAILURE"
260 CALL XDBEXPLAIN(S[*])
270 GOTO 290
280 PRINT "YOUR PASSWORD DOES NOT GRANT ACCESS TO DATA SET"
290 GOSUB 9000
300 GOTO 9900
310 PRINT "NO SUCH SUPPLIER"
320 GOTO 430
330 CALL XDBDELETE(B$,D1$,FNI(1),S[*])
340 IF S[1]<>44 THEN GOTO 370
```

BASIC

```
 350 PRINT "INVENTORY ENTRIES EXIST,SUPPLIER CANNOT BE DELETED"
 360 GOTO 430
 370 IF S[1]=0 THEN GOTO 420
 380 IF S[1]=-23 THEN 280
 390 PRINT "DBDELETE FAILURE"
 400 CALL XDBEXPLAIN(S[*])
 410 GOTO 9900
 420 PRINT "SUPPLIER DELETED"
 430 GOSUB 9000
 440 GOTO 110
9000 CALL XDBUNLOCK(B$,"",M1,S[*])
9010 IF S[1]=0 THEN RETURN
9020 PRINT "DBUNLOCK FAILURE"
9030 CALL XDBEXPLAIN(S[*])
9040 GOTO 9900
9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBEXPLAIN(S[*])
9320 STOP
9900 CALL XDBCLOSE(B$,"",FNI(1),S[*])
9910 IF S[1]=0 THEN STOP
9920 PRINT "DBCLOSE FAILURE"
9930 GOTO 9310
9999 END
```

In the example above, the program calls XDBLOCK to lock the SUP-MASTER data set. Since mode 4 is used, the program must check the condition word when DBLOCK returns control to verify that the data base is locked and the calling program has exclusive access. If this is so, the condition word is 0.

If the data is successfully locked, the program performs the necessary data base operations. In this case, it deletes an entry. Before the entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling XDBGET. The program may request the name of the supplier whose record is to be deleted and use XDBGET in calculated mode to locate the appropriate entry. If entries in the INVENTORY data set exist that have the same SUPPLIER value as the entry to be deleted, the condition word is set to 44 and the entry is not deleted.

After the entry is deleted the data set is unlocked by XDBUNLOCK.

A null list can be used with DBGET to locate an entry to be deleted.

REQUEST DATA SET INFORMATION

```
 10 DIM B$[8],P$[8]
 20 INTEGER S[10],D2[7],M
 30 B$="  STORE;"
 40 INPUT "ENTER PASSWORD: ",P$[1;8]
 50 INPUT "ENTER ACCESS MODE (1-8): ",M
 60 CALL XDBOPEN(B$,P$,M,S[*])
 70 IF S[1]<>0 THEN 9300
300 M=203
310 CALL XDBINFO(B$,"",M,S[*],D2[*])
320 IF S[1]=0 THEN 350
330 CALL DBEXPLAIN(S[*])
340 GOTO 9900
350 PRINT "YOU HAVE ACCESS TO";D2[1];"DATA SETS AS FOLLOWS:"
360 FOR I=2 TO D2[1]+1
370   PRINT D2[I]
380 NEXT I
390 GOTO 9900
         •
```

9300 (same as XDBOPEN example)

9900 (close data base)

The procedure call in this example obtains the numbers of data sets that are available to the current user class by specifying mode 203. If the user class number is 12 and the procedure executes successfully, the D2 array contains:

D2(1)	4	Access to 3 data sets.
D2(2)	2	Read access to data set 2.
D2(3)	–3	Modify access to data set 3
D2(4)	–5	and data set 5.
D2(5)	6	Read and possibly update access to data set 6.

BASIC

REWIND DATA SET

```
 10 DIM B$[8],P$[8],D1$[14],L1$[20],S1$[16],S2$[2]
 20 INTEGER S[10],M,M1,M2
 30 B$="  STORE;"
 40 M1=1
```

(open data base)

```
210 D1$="SUP-MASTER;"
        •
```

(read data set serially)

```
900 INTEGER M3
910 M3=3
920 CALL XDBCLOSE(B$,D1$,M3,S[*])
930 IF S[1]=0 THEN RETURN
940 PRINT "DBCLOSE FAILURE"
950 CALL XDBEXPLAIN(S[*])
960 GOTO 9900
```

9900 (close data base)

To rewind the SUP-MASTER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the Data Set Control Block for SUP-MASTER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request reads the first entry in the data set.

CLOSE DATA BASE

```
  10 DIM B$[8],P$[8]
  20 INTEGER S[10],M
  30 B$="  STORE;"
  40 DEF INTEGER FNI(X)=X

              •
              •
              •

9900 CALL XDBCLOSE(B$,"",FNI(1),S[*])
9910 IF S[1]=0 THEN STOP
9920 PRINT "DBCLOSE FAILURE"
9930 GOTO 9310
9999 END
```

This call closes the data base. It is issued after the program has completed all data base operations and before program termination.

PRINT ERROR

```
  10 DIM B$[8]
  20 INTEGER S[10]
              •
              •
              •
9310 CALL XDBEXPLAIN(S[*])
9320 STOP
```

A call to DBEXPLAIN prints a message on the $STDLIST device which interprets the contents of the status array, S. This is the routine which is called to display the status in the preceding examples.

MOVE ERROR TO BUFFER

```
  10 DIM B$[8],P$[8],M$[72]
  20 INTEGER S[10],M,M1
  30 B$="  STORE;"
  40 M1=1
  50 INPUT "ENTER PASSWORD: ",P$[1;8]
  60 INPUT "ENTER ACCESS MODE (1-8): ",M
  70 CALL XDBOPEN(B$,P$,M,S[*])
  80 IF S[1]<>0 THEN 9300
  90 PRINT "DATA BASE OPENED"
 100 GOTO 9900
9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBERROR(S[*],M$)
9320 PRINT M$
9330 STOP
```

In this example, a call to DBERROR returns one of the messages appropriate to the current condition word. For example, if the condition word is equal to 16, the message returned in M$ is THE DATA SET IS FULL. Note that the length parameter need not be included since the logical length of M$ is set by XDBERROR.

RPG EXAMPLES

The following restrictions apply to IMAGE data bases used with RPG:

1. Data is added and retrieved as complete entries, in other words, you cannot read or modify single items through RPG. Therefore, if the RPG program is to read an entry from a data set, the specified password must correspond to a user class number allowing read access to all data items in the entry. If the RPG program is to write an entry to a data set, the password must correspond to a user class number allowing write access to all data items in the entry.

   Since entries are handled in this way, data sets to be used with RPG programs are sometimes defined with one-item entries. However, if you intend to use QUERY with the data set you may need to define more items.

2. Only one search item can be used to reference a data set in a program unless the data set is defined as more than one file, or you are doing an ISAM simulation and processing between limits. (Consult the *RPG/3000 Compiler Reference and Application Manual* for more information.)

3. RPG supports all the DBGET procedure input modes except reread. It provides two additional modes:

   - simulated indexed sequential read, forward and backward

   - read down chain until key changes.

RPG PROGRAMS AND IMAGE

To use an IMAGE data base through RPG application programs you must describe the data base with file description specifications. A data set may be described by more than one file description specification to allow you to access it in more than one way, for example, performing both serial and chained reads or using two different search items (keys). The file description specification and its continuation records specify:

- an IMAGE file by naming both the data base and a data set within it

- a search item name

- an access mode (1 through 8)

- a password

- an input/output mode for the file.

In addition, you can add and delete entries with special RPG output specifications.

Complete instructions for using an IMAGE data base through RPG programs are given in the *RPG/3000 Compiler Reference and Application Manual.* The RPG program must be executed from the account and group which contains the data base.

Note that RPG programs automatically lock the whole data base when it is opened with access mode 1. Locking and unlocking are part of the RPG program cycle described in the RPG manual.

Figure 5-7 contains a sample RPG program which reads the SALES entries associated with a particular stock number and prints the contents in a report. The file description specifications include:

- line 0003 — a description of the SALES data set as a chained input file with fixed length records 38 bytes long. The processing mode used for the data set is random. The key field is 8 bytes long and contains alphanumeric data. The file organization code M signifies an IMAGE file. The file name is a logical data set name, in other words, it can be a reminder of the actual data set name (see discussion of line 0007 below.)

- line 0004 — a data base name record specifying the STORE data base, an access (open) mode of 3, and input/output mode C (chained sequential read),

- line 0005 — an item name record specifying the STOCK# search item as the key,

- line 0006 — a level identification record specifying the DO-ALL password,

- line 0007 — a data set name record specifying the SALES data set. This is the actual data set name and overrides the file name, which may be a logical name identifying the data set. Since RPG file names cannot exceed 8 characters and can contain no special characters, a file specification for the SUP-MASTER data set or DATE-MASTER data set should have file names such as SUP and DATE with the full names given in a data set name record.

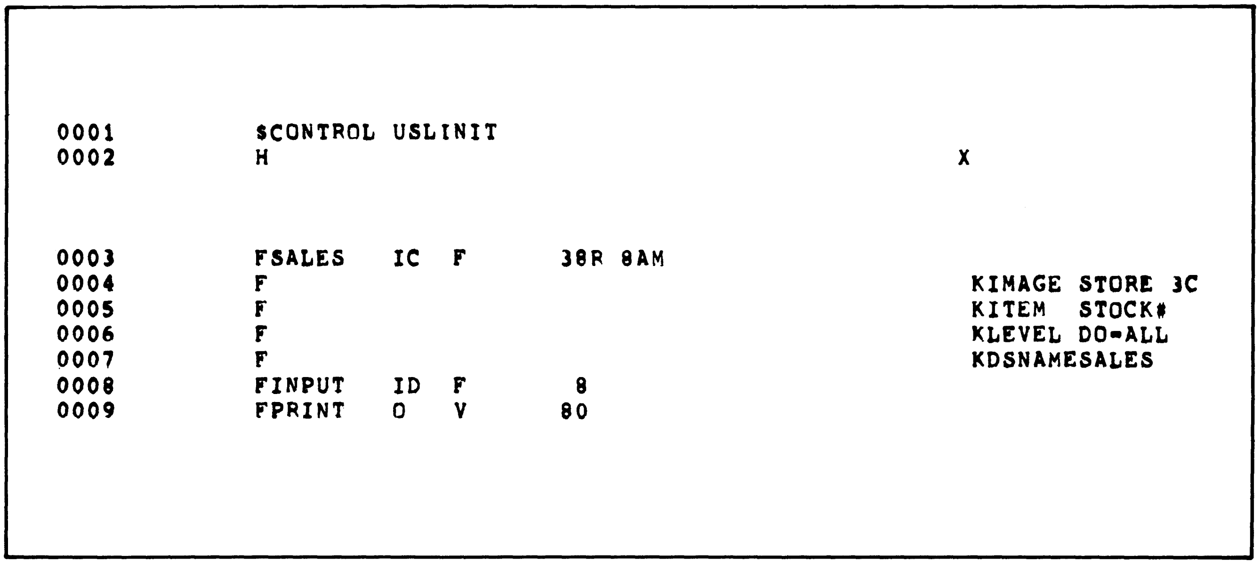
- line 0008 — a description of the INPUT file as a demand file with fixed length records 8 bytes long.

- line 0009 — a description of the PRINT file as an output file of variable length records which are at most 80 characters long.

The input specifications describe:

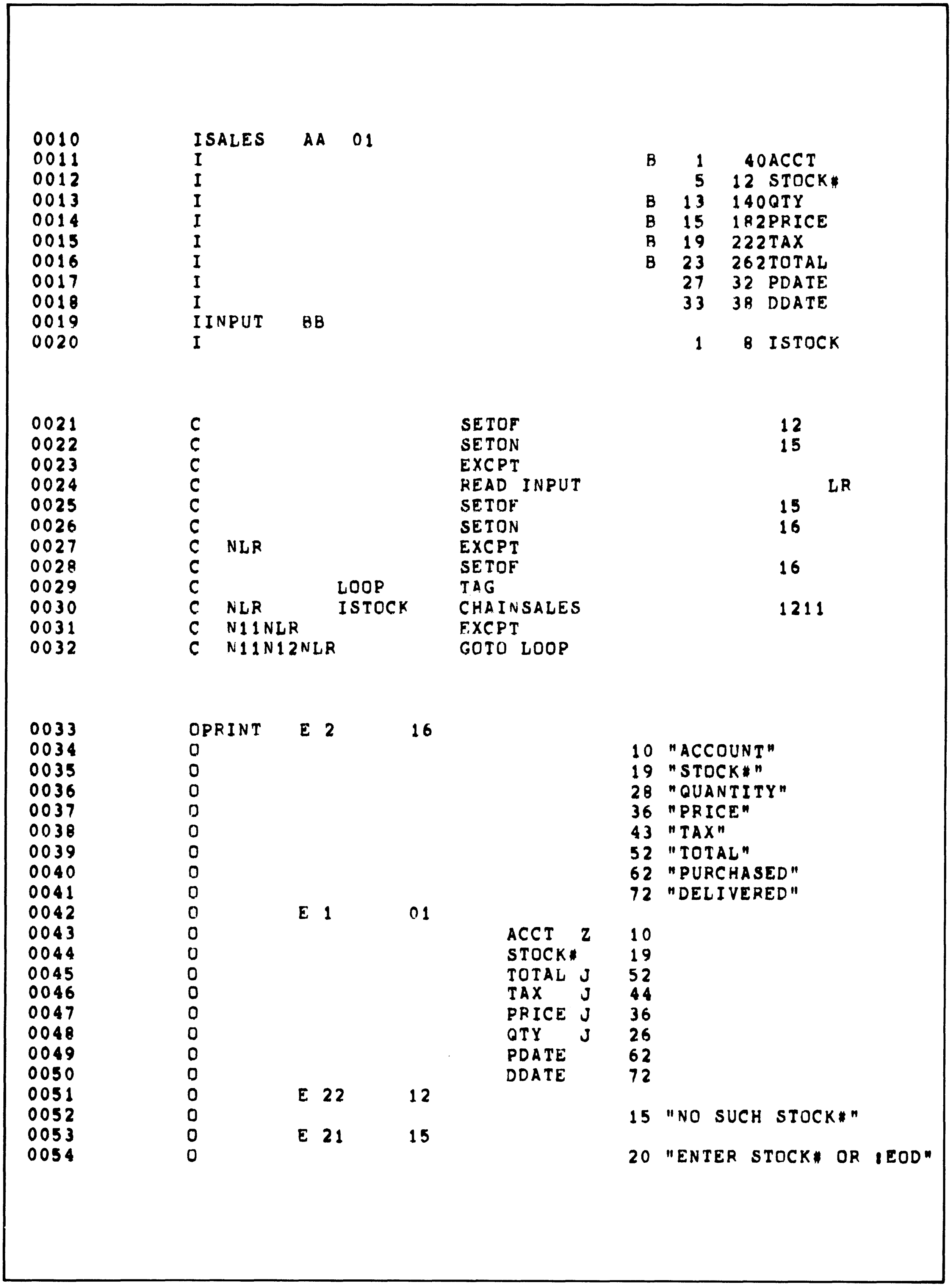
- line 0010 through 0018 — a SALES data entry with five binary and three character (ASCII) data items,

- lines 0019 through 0020 — an INPUT record of 8 bytes with a field named ISTOCK.



```
0001         $CONTROL USLINIT
0002         H                                            X


0003         FSALES  IC   F      38R 8AM
0004         F                                             KIMAGE STORE 3C
0005         F                                             KITEM  STOCK#
0006         F                                             KLEVEL DO-ALL
0007         F                                             KDSNAMESALES
0008         FINPUT  ID   F        8
0009         FPRINT  O    V       80
```

Figure 5-7. Sample RPG Program



```
0010         ISALES   AA  01
0011         I                                    B   1   40ACCT
0012         I                                        5  12 STOCK#
0013         I                                    B  13  140QTY
0014         I                                    B  15  182PRICE
0015         I                                    B  19  222TAX
0016         I                                    B  23  262TOTAL
0017         I                                       27  32 PDATE
0018         I                                       33  38 DDATE
0019         IINPUT   BB
0020         I                                        1   8 ISTOCK


0021         C                     SETOF                     12
0022         C                     SETON                     15
0023         C                     EXCPT
0024         C                     READ INPUT                    LR
0025         C                     SETOF                     15
0026         C                     SETON                     16
0027         C  NLR                EXCPT
0028         C                     SETOF                     16
0029         C           LOOP      TAG
0030         C  NLR      ISTOCK    CHAINSALES                1211
0031         C  N11NLR             EXCPT
0032         C  N11N12NLR          GOTO LOOP


0033         OPRINT   E 2      16
0034         O                                   10 "ACCOUNT"
0035         O                                   19 "STOCK#"
0036         O                                   28 "QUANTITY"
0037         O                                   36 "PRICE"
0038         O                                   43 "TAX"
0039         O                                   52 "TOTAL"
0040         O                                   62 "PURCHASED"
0041         O                                   72 "DELIVERED"
0042         O        E 1      01
0043         O                         ACCT  Z   10
0044         O                         STOCK#    19
0045         O                         TOTAL J   52
0046         O                         TAX   J   44
0047         O                         PRICE J   36
0048         O                         QTY   J   26
0049         O                         PDATE     62
0050         O                         DDATE     72
0051         O        E 22     12
0052         O                                   15 "NO SUCH STOCK#"
0053         O        E 21     15
0054         O                                   20 "ENTER STOCK# OR :EOD"
```

Figure 5-7. Sample RPG Program (Continued)

The calculation specifications, lines 0021 through 0032 request input to the INPUT file which can be equated to $STDIN by using the MPE :FILE command before executing the program. They also read the SALES entries with values equal to the stock number entered, print the information, and, when the end of chain is encountered, request another stock number.

The output specifications, lines 0033 through 0050 describe a report with column headings for each item and one line records for each entry. The ACCT item is edited with a Z edit specification and the QTY, PRICE, TAX, and TOTAL items with a J edit specification.

The last output specifications, lines 0051 through 0054, describe the message to be printed if there is no entry with the requested stock number value and the message which prompts for the stock number.

CREATING AND MAINTAINING THE DATA BASE

SECTION VI

The IMAGE utility programs create and initialize the data base files and perform various maintenance functions. The programs are DBUTIL, DBSTORE, DBRESTOR, DBLOAD, and DBUNLOAD. Here is a brief summary of the utility routines and their functions:

- DBUTIL. The DBUTIL program performs several different functions corresponding to the following commands:

  HELP — This command provides a description of all other DBUTIL commands.

  CREATE — Before data can be entered in and retrieved from the data base, this command must be used to create and initialize a data base file for each data set.

  ERASE — This command erases existing data entries from all data sets and returns the data base to the state it was in after DBUTIL CREATE was executed. It should be used before loading data entries that have been stored using the DBUNLOAD utility program back into the data base.

  PURGE — With this command, you can purge the entire data base including the root file and all data set files. This routine should be used before restoring the data base when performing recovery operations and before creating a new version of the data base while restructuring it.

  ACTIVATE — This command prepares a data-base-access file to be used when you are accessing a data base that resides on a remote HP 3000 computer.

  DEACTIVATE — To deactivate a data-base-access file before making changes to it, use this command.

  VERIFY — To determine whether or not a particular data-base-access file is activated or deactivated, use this command. If deactivated, the content is checked to see if the file is a data-base-access file.

  SET — This command may be used to change or remove the maintenance word, or to specify the number of buffers to be used on the basis of the number of users accessing the data base.

  SHOW — With this command, information about the data base may be displayed on the terminal or line printer. It is intended for use by the data base manager in performing data base maintenance functions.

  EXIT — This command terminates DBUTIL program execution.

  To maintain compatibility with earlier versions of DBUTIL, the CREATE, ERASE, and PURGE commands may also be executed by specifying them as DBUTIL entry points.

- DBSTORE. It is important to keep a backup copy of the data base in order to restore it should some hardware or operating system failure occur. This program copies the entire data base including the root file and all data sets to magnetic tape or serial disc volumes in a format compatible with the output created by the MPE :STORE and :SYSDUMP commands. Backup procedures should be established to execute this program on a regular basis.

- DBRESTOR. If the data base must be restored, you can use this program to copy to disc the data base from magnetic tape or serial disc volumes created by the DBSTORE program or the MPE :STORE or :SYSDUMP commands. Prior to executing this routine you should use the DBUTIL PURGE command to remove the existing data base files from the group and account.

- DBUNLOAD. This program copies all the data entries of each data set to specially formatted magnetic tape or serial disc volumes. It is particularly useful if you want to modify the data base structure slightly, for example, increase the capacity of a data set. In this case, you purge the data base, change the schema and create a new root file, execute DBUTIL CREATE, and then reload the data entries from the volumes created by DBUNLOAD.

  DBUNLOAD arranges the data entries in each set placing the chained entries of the primary path in contiguous order. After the data base is reloaded, chained access along the primary path is more efficient.

- DBLOAD. With this program you can load the data entries copied by DBUNLOAD back into the data sets. If the data base has not been recreated, the DBUTIL ERASE command should be executed prior to using DBLOAD.

USING THE UTILITIES TO RESTRUCTURE THE DATA BASE

It is possible to make certain changes to the design of an existing data base without having to write special programs to transfer data from the old data base to the new one. The general sequence of operations which you use to do this is:

1. Run DBUNLOAD on the old data base, copying all the data entries to tape or serial disc.
2. Purge the old data base using DBUTIL,PURGE.
3. Redefine the data base using the same data base name and create a new root file with the Schema Processor.
4. Use the DBUTIL CREATE command to create and initialize the data sets of the new data base.
5. Run DBLOAD on the new data base using the tape or serial disc created in step 1 to put the old data into the new base.

DESIGN CHANGES

The data base design changes for which the above procedure functions correctly are limited. Schema changes that yield correctly transformed data bases fall into two general categories: those which always result in a good transformation and those which are legitimate only in some circumstances.

Any of the following schema changes, alone or combined, which are acceptable to the Schema Processor will always result in a successfully transformed data base:

- Adding, changing, or deleting passwords and user class numbers
- Changing a data item or data set name and all references to it
- Changing data item or data set read and write class lists
- Adding new data item definitions
- Removing or changing definitions of unreferenced data items
- Increasing data set capacities
- Adding deleting, or changing sort item designators

- Changing primary paths
- Adding new data items to the original end of a data entry definition
- Removing data items from the original end of a data entry definition
- Changing an automatic master to a manual master or vice versa.

Other schema changes may or may not be legitimate. A potential change must be judged in light of the particular data base and the functioning of DBUNLOAD and DBLOAD, described later in this section. Basically, all entries from an old data set are put into the new data set with the same number, except that no entries are directly put into automatic masters. The entries are truncated or padded with zeroes as necessary to fit the new data set's entry length. DBUNLOAD and DBLOAD always handle full entries, without regard to item positions or lengths. If the new data set's entry is defined with the items in a different order than the old data set, DBLOAD will not fail but the data set content may nevertheless be invalid. For example, data of type real may now occupy the position of a character type item.

In some circumstances, DBLOAD will fail. For example, if a data set's capacity has been reduced in the new data base to a number less than the number of that data set's entries on the tape or serial disc.

BACKUP AND RECOVERY

The IMAGE software is designed to maintain and ensure the integrity of IMAGE data bases. There is the possibility, however, of losing data or data base structure information due to hardware failure, operating system crash, or some other external cause. It is, therefore, prudent to adopt backup procedures of one type or another in anticipation of possible trouble. The data base manager must be aware also of the system manager backup and recovery activity since the data base files may be affected by it.

NOTE

If an operating system crash occurs, you must do a complete recovery for any data base that was open at the time the crash occurred. A COOLSTART is not sufficient since damage to chain information is not detected and the count of chain entries may not be accurate in the files.

Three different approaches to data base backup and recovery are summarized below. Each approach requires the absolute cooperation of all operations personnel.

METHOD 1

One approach is to depend on a daily system dump and system reload, if needed, by operations personnel. This solution is a viable one if all the following conditions are true:

- You are able to recover from a midday system failure by rerunning all jobs that modified the data base.
- You are immediately informed of all system reloads so that you will not run additional jobs prior to bringing the reloaded data base up-to-date.
- You have access to the system dump output, so that you may restore the data base if needed after a system crash which is not followed by a system reload.

METHOD 2

A second approach is for you to maintain your data base independently. At regular intervals you must copy the data base using DBSTORE. After a system crash, you must restore the data base using DBRESTOR. You may want to use this method for the following reasons:

- System crashes are not always followed by reloads, therefore, this method ensures the integrity of your data base.
- It may be desirable to maintain more recent copies of the data base than can be provided by the system backup procedures in order to minimize the number of transactions lost after a crash.

METHOD 3

A third approach, where multiple data bases and files are involved, is to have them all within one group or account and, having the account manager capability, utilize the STORE command to copy them collectively to tape or serial disc and, if necessary, to restore them collectively with the RESTORE command.

RECOVERING CHANGES MADE AFTER BACKUP

Each of the three methods described above merely restores the data base to its state at backup time. No automatic recovery is provided to redo changes made between the time the backup copy of the data base is created and the crash occurs. This problem can be minimized by a combination of scheduling and sensible restrictions on the concurrent updating of common data bases by more than one batch or session application.

Transactions logged by an application which is changing the data base can then be used to reinstate the data base to its exact state at the time of a crash. A difficult situation for which to provide backup is one in which many applications can concurrently modify a single data base. In such cases it may be acceptable to store the data base more frequently and not to attempt dynamic recovery. In any case, your installation must design backup and recovery procedures to meet your particular needs.

SALVAGING DATA

If a data base structure is damaged, and no backup copies are available, it may be possible to salvage most or all of the data by serially reading the data entries, writing them to a tape or disc file, recreating the data base, and reloading the data. If structure damage is detected by an abnormal termination of the DBUNLOAD program running in CHAINED mode, or by a discrepancy between the number of entries unloaded and the number expected from one or more data sets, it may be possible to unload the data base by running DBUNLOAD in SERIAL mode, which does not depend on internal linkages. These DBUNLOAD modes are discussed later in this section.

If all necessary existing manual master data entries are written to tape or serial disc, reloading the data base using the DBLOAD program, after erasing the data base using DBUTIL, results in a structurally intact approximation of the original data base.

ACTIVATING AND DEACTIVATING A DATA-BASE-ACCESS FILE

The DBUTIL program provides two commands, ACTIVATE and DEACTIVATE, for use in relation to remote data base access. The various methods for accessing a data base on a remote HP 3000 are described in Section VIII and you should read that section before using the DBUTIL ACTIVATE and DEACTIVATE commands. These commands allow you to activate and deactivate a data-base-access file to be used with the third remote-data-base access method described in Section VIII.

UTILITY PROGRAM OPERATION

The utility programs may be run in either job or session mode, but you must log on with the account and group that contains the root file. Therefore, you can not use the utility programs with a remote data base unless you initiate a remote session and run the utility program as part of that session. The IMAGE utility programs do not allow you to use the :FILE command to equate a data base or data-base-access file name to a different file.

To execute the DBUTIL CREATE command or to change or remove the maintenance word with the DBUTIL SET command, you must also log on with the same user name that was used when the Schema Processor created the root file; this verifies to IMAGE that you are the data base creator. To operate the other utility routines and enter other DBUTIL commands, you need not be the data base creator. However, in order to perform some utility functions you must know the maintenance word if the data base creator has defined one. If no maintenance word is defined, only the data base creator can execute the other utility programs and the DBUTIL commands that require a maintenance word.

BACKUP FILES

The backup files created by DBSTORE and DBUNLOAD may be written to magnetic tape or serial disc volumes. In the discussion of the utility programs that follows, the term *volume* refers to either a magnetic reel or a serial disc pack.

ERROR MESSAGES

Some of the error messages are described with the operating instructions for the utility programs. Appendix A contains a complete summary of the error messages issued by these programs.

DBUTIL

The DBUTIL program performs several different functions according to the command you enter. Each DBUTIL command is described separately on the following pages.

OPERATION

① :RUN DBUTIL.PUB.SYS

② >> *command*

1. Initiates execution of the DBUTIL program which is in the PUB group and SYS account.

2. Prompts for a DBUTIL *command*. Enter one of the following:

   HELP
   CREATE
   ERASE
   PURGE
   DEACTIVATE
   ACTIVATE
   VERIFY
   SET
   SHOW
   EXIT

   DBUTIL commands may be abbreviated to the first three characters. For example, CREATE may be abbreviated to CRE.

When using the CREATE, PURGE, or ERASE commands, you can bypass the command prompt by specifying the full command as an entry point with the :RUN command; for example, :RUN DBUTIL.PUB.SYS,CREATE. If you use an entry point, IMAGE prompts you for the data base name and, optionally, for the maintenance word.

Data base name: *data base name* [*/maintenance word*]

data base name — is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word — is an optional ASCII string, one to eight characters long with no commas or semicolons, which defines a password to be used by anyone other than the data base creator to enable them to execute certain DBUTIL commands, and operate the other utility programs.**

In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.

Note that to perform any DBUTIL command except SHOW, HELP, or EXIT, you must have exclusive access to the data base or data-base-access file.

**The data base creator may also define or change the maintenance word by using the SET command.

ACTIVATE

Activates the data-base-access file for use with DBOPEN. Before using this command, read the description of remote data base access in Section VIII.

This command should be used to prepare a data-base-access file before accessing a remote data base residing on another HP 3000. IMAGE checks the content of the file to assure that the FILE equation includes the DEV= parameter, the DSLINE command contains the same dsdevice referenced in the FILE equation, and that record 3 and all subsequent records include =HELLO and do not contain a DSLINE parameter.

> The form of the ACTIVATE command is
>
> ACT[IVATE] *data-base-access file name*

For example,

```
ACTIVATE XXXDBA
            ↖
             data-base-access file name
```

where

data-base-access file name is the name of the data-base-access file that you created with the Editor.

The data-base-access file may have the same name as the data base or it may have another name, but the name must conform to the rules for data base names.

If DBUTIL successfully activates the file, it prints a confirmation message on the *listfile* device.

UNEXPECTED RESULTS

The following message is displayed if the specified file is not the correct structure or format:

Message	Meaning
Invalid contents of ascii access file	Record 1 does not contain FILE command with DEV=dsdevice parameter, Record 2 does not contain DSLINE command with same dsdevice as Record 1, or one of remaining records does not contain =HELLO, or does contain a DSLINE parameter.

Refer to Appendix A for other error messages.

EXAMPLE (Session Mode)

```
:RUN DBUTIL.PUB.SYS                 Initiate DBUTIL execution.
      .
      .
>>ACT XXXDBA                        Enter abbreviated form of ACTIVATE
  Data-base-access file             command and data-base-access file name.
      XXXDBA is ACTIVATED
>>
```

DBUTIL checks the structure of the file named XXXDBA for correct format and activates the file. You will not be able to edit the file unless you deactivate it using the DBUTIL DEACTIVATE command.

CREATE

Creates and initializes a file for each data set in the data base.

DBUTIL initializes each data set to zeroes and saves it as a catalogued MPE file in the same log on group as the root file. The data set file names are created by appending two digits to the root file name. If the root file is named XXXX, then the first data set defined in the schema is in a file named XXXX01, the second data set file is named XXXX02, and so forth.

To execute the DBUTIL program to create and initialize the data base you must be the data base creator; that is, you must log on with the same user name, account and group that was used to run the Schema Processor and create the root file.

> The form of the CREATE command is
>
> CRE[ATE] *data base name*[*/maintenance word*]

For example,

```
        CREATE STORE/XYZED
                  /     \
 data base name          maintenance word
```

where

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word is an optional ASCII string, one to eight characters long with no commas or semicolons, which defines a password to be used by anyone other than the data base creator to enable them to execute certain DBUTIL commands and operate the other utility programs.

After DBUTIL has created and initialized the data base files, it prints a confirmation message on the *listfile* device and prompts for another command.**

EXAMPLE (Session Mode)

```
:RUN DBUTIL.PUB.SYS
          .
          .
>>CREATE  STORE/XYZED
  Data base STORE has been CREATED
>>
```

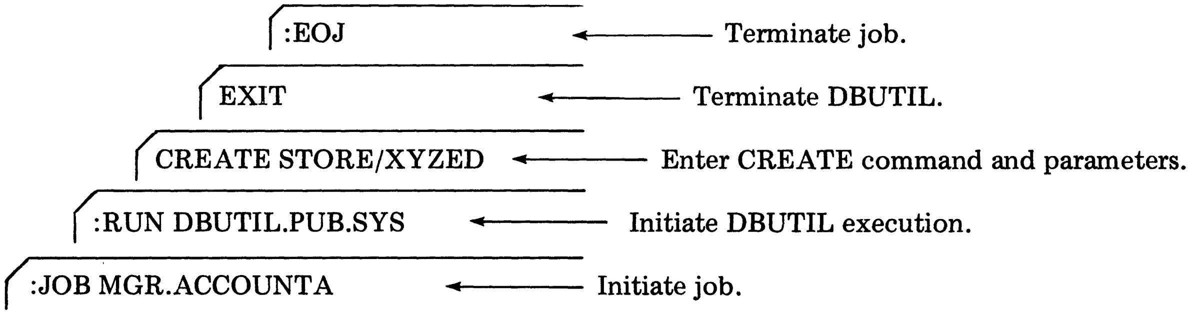
:RUN DBUTIL.PUB.SYS — Initiate DBUTIL execution.

>>CREATE STORE/XYZED — Respond to DBUTIL prompt with CREATE command, data base name, and maintenance word.

DBUTIL creates, initializes, and saves files named STORE01, STORE02, and so forth, one file for each data set. These constitute the empty data base.

**The data base creator may also define or change the maintenance word by using the SET command.

Job mode:



```
:EOJ                       ←——— Terminate job.
EXIT                       ←——— Terminate DBUTIL.
CREATE STORE/XYZED         ←——— Enter CREATE command and parameters.
:RUN DBUTIL.PUB.SYS        ←——— Initiate DBUTIL execution.
:JOB MGR.ACCOUNTA          ←——— Initiate job.
```

After the data files are created and initialized, DBUTIL prints the message:

```
DATA BASE STORE HAS BEEN CREATED
```

on the *listfile* device.

DEACTIVATE

Deactivates the data-base-access file to disallow access or to allow modification to be made to the file.

This command should be used before you change the contents of the data-base-access file. (Refer to Section VIII for more information about accessing remote data bases.)

> The form of the DEACTIVATE command is
>
> DEA[CTIVATE] *data-base-access file name*

For example,

```
DEACTIVATE XXXDBA
                ↖
                 data-base-access file name
```

where

data-base-access file name is the name of the data-base-access file to be deactivated.

If DBUTIL successfully deactivates the file, it prints a confirmation message on the *listfile* device.

EXAMPLE (Session Mode)

```
:RUN DBUTIL.PUB.SYS
         .
         .
>>DEA   XXXDBA
  Data-base-access file
       XXXDBA is DEACTIVATED
>>
```

Initiate DBUTIL execution

Enter an abbreviated DEACTIVATE command and the data-base-access file name.

ERASE

Reinitializes the data sets to their empty condition.

The data sets remain as catalogued MPE files. To execute DBUTIL to reinitialize the data sets you must be the data base creator or supply the correct maintenance word. This utility function should be performed before data saved by DBLOAD is loaded back into the data base unless it was recreated.

> The form of the ERASE command is
>
> ERA[SE] *data base name* [*/maintenance word*]

For example,

```
          ERASE STORE/XYZED
                  ↗    ↖
 data base name          maintenance word
```

where

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word is the maintenance word defined by the data base creator when the data base is created with DBUTIL. This word must be supplied by anyone other than the data base creator.

After DBUTIL has completely reinitialized the data sets, it prints a confirmation message on the *listfile* device.

EXAMPLE (Session Mode)

```
:RUN DBUTIL.PUB.SYS                     Initiate DBUTIL execution.
          .
          .
>>ERASE STORE/XYZED                     Enter ERASE command, data base name,
  Data base STORE has been ERASED       and maintenance word.
>>
```

DBUTIL reinitializes the data set files STORE01, STORE02, and so forth to their original empty, zeroed condition.

EXIT

Terminates DBUTIL execution.

The form of the EXIT command is

```
EXI[T]
```

EXAMPLE (Session Mode)

```
>>CREATE STORE
  Data base STORE has been CREATED
>>EXIT

 END OF PROGRAM
```

Create a data base.

If no other DBUTIL functions are to be performed, terminate DBUTIL with EXIT command.

HELP

Displays all DBUTIL commands.

The form of the HELP command is

```
HEL[P] [commandname]
```

For example,

```
HELP  ERASE
        ↖
          commandname
```

where

commandname is the name of a specific DBUTIL command whose format you want to display. The name may be abbreviated to the first three letters.

If you do not specify a *commandname*, the HELP command lists the names of all valid DBUTIL commands.

If you specify a *commandname*, the correct syntax for that command is displayed.

EXAMPLES

```
>>HELP

  HELP          CREATE       ERASE       PURGE        DEACTIVATE

  ACTIVATE      VERIFY       SET          SHOW            EXIT

>>HELP CREATE

         CRE[ATE] data base name[/maintenance word]

>>
```

PURGE

Purges the root file and all the data sets of the referenced data base.

Purging removes the files from the catalog and returns the disc space to the system. As with ERASE, you must be the data base creator or must provide the maintenance word to use DBUTIL with the PURGE entry. Before running the DBRESTOR program to restore a data base, you should use this utility function to purge the data base.

> The form of the PURGE command is
>
> PUR[GE] *data base name* [*/maintenance word*]

For example,

```
          PURGE STORE/XYZED
                  ↗     ↖
data base name            maintenance word
```

where

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word is the maintenance word defined by the data base creator when the data base is created with DBUTIL. This word must be supplied by anyone other than the data base creator.

If DBUTIL successfully purges the data base, it prints a confirmation message on the *listfile* device.

UNEXPECTED RESULTS

The following messages are printed if an unexpected situation occurs:

Message	Meaning
No root file, PURGE operation proceeding	DBUTIL was unable to locate the root file, but will attempt to purge any data set files.
Data set XXX*k* has been purged	DBUTIL successfully purged the root file and the *n* data sets of the data base. However, DBUTIL also discovered and purged an unexpected data set named XXXX*k* where *k* is a number greater than the number of data sets defined for the data base (*n*).
Data set XXXX*k* is missing	DBUTIL successfully purged the root file and all existing data sets but data set XXXX*k* is unexpectedly missing. In this case *k* is less than the number of data sets defined for the data base.
Data base PURGE is not complete	A fatal error occurred while DBUTIL was attempting to purge the data base. The fatal error message is printed above this one. Some of the data sets have been purged.

Refer to Appendix A for other error messages.

EXAMPLE (Session Mode)

```
:RUN DBUTIL.PUB.SYS                    Initiate DBUTIL execution.
        .
        .
>>PURGE  STORE                         Enter PURGE command and data base name
  Data base STORE has been PURGED      (assuming there is no maintenance word).
```

DBUTIL confirms that the user is logged on with the same user name, account, and group which were used to create the data base. It then determines whether the root file exists and if so, purges the root file and any files named STORE01, STORE02, and so forth. Even if the root file does not exist, any data set files with names based on the root file name are purged.

DBUTIL

SET

Changes or removes the maintenance word or specifies the number of input/output buffers to be allocated by IMAGE in the DBCB depending on the number of users concurrently accessing the data base. Only the data creator can change or remove the maintenance word.

The form of the SET command is

```
                                              ⎧MAINT = maintenance word                              ⎫
SET data base name [/maint word]              ⎨BUFFSPECS = num buffers (from-users/to-users)         ⎬
                                              ⎩         [,num buffers (from-users/to-users)] . . .   ⎭
```

For example,

```
SET DBINVEN  MAINT=ZZZZZ
```

data base name

maintenance word

or

from-users

```
SET STORE/XYZED BUFFSPECS = 4(1/5),7(6/10),9(11/15),10(16/20)
```

data base name

num buffers

to-users

maint word

where

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maint word is the current maintenance word for the data base, and must be supplied by anyone other than the data base creator.

maintenance word is the new maintenance word for the data base. If omitted, the currently defined maintenance word is removed and the data base has no maintenance word. Only the data base creator can change or remove the maintenance word.

num buffers is the number of buffers to be allocated by IMAGE in the DBCB for the range of users specified between the parentheses that follow. The minimum number of buffers allowed is 4 and the maximum is 255.

from-users is the minimum number of concurrent users (access paths) for which the preceding *num buffers* should be allocated. The minimum *from-users* value allowed is 1 and the maximum is 120. The value must be greater than the immediately preceding *to-users* value.

to-users is the maximum number of concurrent users for which the preceding num buffers should be allocated. The minimum *to-users* value allowed is 1 and the maximum 120. The value must be greater than the immediately preceding *from-users* value.

The *from-users/to-users* ranges must be specified in increasing order. The ranges may not overlap but they need not be consecutive. If *num buffers* is not specified for a particular number of users, the default number of buffers is used. These are the default settings assigned by IMAGE:

b(1/2)
b+1(3/4)
b+2(5/6)
b+3(7/8)
b+4(9/10)
b+5(11/12)
b+6(13/14)
b+7(15/16)
b+8(17/18)
b+9(19/20)

The value of b is equal to

- the largest number of search items in any detail data set in the data base plus 3,
- or 8,

whichever is larger. If p is the maximum number of search items (the path count), the value of b can be represented as b = max (p+3,8).

For example, the largest path count for a detail data set in the store data base is 4. (This is the path count for the SALES data set.) Therefore, the value of b for the STORE data base is:

b = max (4+3,8) = 8

The default buffer specifications in this case are:
8(1/2),9(3/4),10(5/6),11(7/8),12(9/10),13(11/12),14(13/14),15(15/16),16(17/18),17(19/120).

EXAMPLES (Session Mode)

```
:RUN DBUTIL.PUB.SYS                          Initiate DBUTIL execution.
           •
           •
>>SET XYZDB MAINT=                           Remove current maintenance word.
  Maintenance word changed
>>
:RUN DBUTIL.PUB.SYS
           •
           •
>>SET STORE BUFFSPECS=4(1/120)               Specify 4 buffers to be allocated for
  For data base STORE                        from 1 to 120 users (access paths).

  BUFFER SPECIFICATIONS:                     DBUTIL confirms the specifications by
  4(1/120)                                   listing them.
>>
```

SHOW

Displays information about the data base on a terminal or line printer. The information may include a list of processes that have the data base open, the status of locks in the data base, and the current buffer specifications. This command should be used with care for data base maintenance functions since it obtains exclusive control of the data base for several seconds preventing all other access.

. The form of the SHOW command is

```
                                         ⎧ MAINT     ⎫
                                         ⎪ ALL       ⎪
SHO[W] data base name [/maint word]     ⎨ BUFFSPECS ⎬    [OFFLINE]
                                         ⎪ LOCKS     ⎪
                                         ⎩ USERS     ⎭
```

For example,

```
        SHOW STORE/XYZED  ALL OFFLINE
               ↑     ↖
              /       \
   data base name      maint word
```

where

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maint word is the current maintenance word for the data base. It must be supplied by anyone other than the data base creator.

MAINT displays the maintenance word, if any.

ALL displays all the information provided with MAINT, BUFFSPECS, LOCKS, and USERS.

BUFFSPECS displays the current buffer specifications which may either be the IMAGE default settings or the values specified with the DBUTIL SET command.

LOCKS displays the status of locks currently obtained (or requested). (Refer to examples below.)

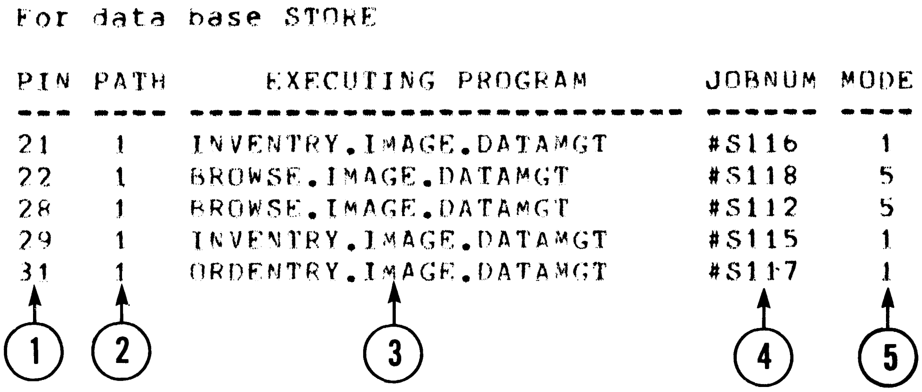
USERS displays a list of the processes that have the data base open with the program file name and other information. (Refer to examples below.)

OFFLINE requests that the information be listed on the line printer. The formal file designator for the list file is DBUTLIST.

The SHOW commands may be executed at any time except when another process has the data base opened in an exclusive access mode (mode 3 or 7).

EXAMPLE (List Users, Session Mode)

```
    :RUN DBUTIL.PUB.SYS
               •
               •
    >>SHOW STORE USERS
```



```
For data base STORE

PIN  PATH      EXECUTING PROGRAM            JOBNUM  MODE
---  ----  ---------------------------      ------  ----
21    1    INVENTRY.IMAGE.DATAMGT           #S116    1
22    1    BROWSE.IMAGE.DATAMGT             #S118    5
28    1    BROWSE.IMAGE.DATAMGT             #S112    5
29    1    INVENTRY.IMAGE.DATAMGT           #S115    1
31    1    ORDENTRY.IMAGE.DATAMGT           #S117    1
```

The columns of information are as follows:

1. The Process Identification Number (PIN). This is a number assigned to a process by the operating system when the process is created. In this list it indicates that the process has the STORE data base open.

2. The access path number. The access paths for each process are numbered consecutively beginning with 1. (Refer to the discussion of access paths in Section IV.)

3. The name of the program file, its group and account.

4. The number of the job or session in which the process is running.

5. The access mode in which the data base is open.

Note that DBUTIL does not call DBOPEN and therefore it is not listed as an executing program.

EXAMPLE (List Buffer Specifications, Session Mode)

```
                •
                •
  >>SHO STORE BUFFSPECS                     List buffer specifications for STORE data
    For data base STORE                     base.

    BUFFER SPECIFICATIONS:
    4(1/120)
  >>
```

The listing above indicates that the current buffer specifications provide for 4 buffers to be allocated when there are between 1 and 120 concurrent users of the data base.

DBUTIL

FORMAT OF SHOW LOCKS LIST

DBUTIL lists the locking information sequentially by locking level: data base locks followed by data set locks, followed by data entry locks. The names of locked entities (for example, the data base, data set, or lock descriptor for data entries) appear in upper case followed by a list of other processes waiting at that locking level. DBUTIL indicates in lower case the reason each process is waiting.*

If the term (PENDING) appears after a locked entity, it indicates that the lock has been obtained but control cannot be returned to the caller until other locks have been released. The same process identification will appear elsewhere in the list together with an explanation of why it is waiting.

Infrequently, the term (TEMP) may appear. This indicates that IMAGE has placed a temporary lock while processing a lock request. These locks are held very briefly and only under rare circumstances.

The Process Identification Numbers (PINs) and job/session numbers listed are the same as those shown by the MPE commands such as :SHOWJOB and :SHOWQ.

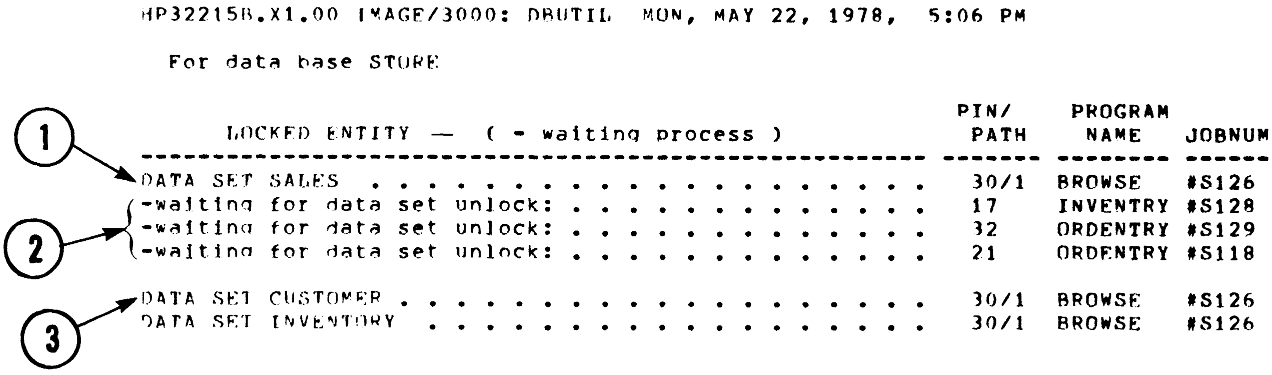
EXAMPLES (List Locks, Session Mode)

•

•

>> SHOW STORE LOCKS OFFLINE — List the status of locks requested and held in the STORE data base on the line printer.

The line printer listing looks like this:



```
HP32215B.X1.00 IMAGE/3000: DBUTIL  MON, MAY 22, 1978,  5:06 PM

  For data base STORE

                                                  PIN/   PROGRAM
        LOCKED ENTITY  --  ( - waiting process )  PATH    NAME    JOBNUM
------------------------------------------------ ------- -------- ------
DATA SET SALES . . . . . . . . . . . . . . . . .  30/1   BROWSE   #S126
-waiting for data set unlock: . . . . . . . . . . 17     INVENTRY #S128
-waiting for data set unlock: . . . . . . . . . . 32     ORDENTRY #S129
-waiting for data set unlock: . . . . . . . . . . 21     ORDENTRY #S118

DATA SET CUSTOMER . . . . . . . . . . . . . . . . 30/1   BROWSE   #S126
DATA SET INVENTORY  . . . . . . . . . . . . . . . 30/1   BROWSE   #S126
```

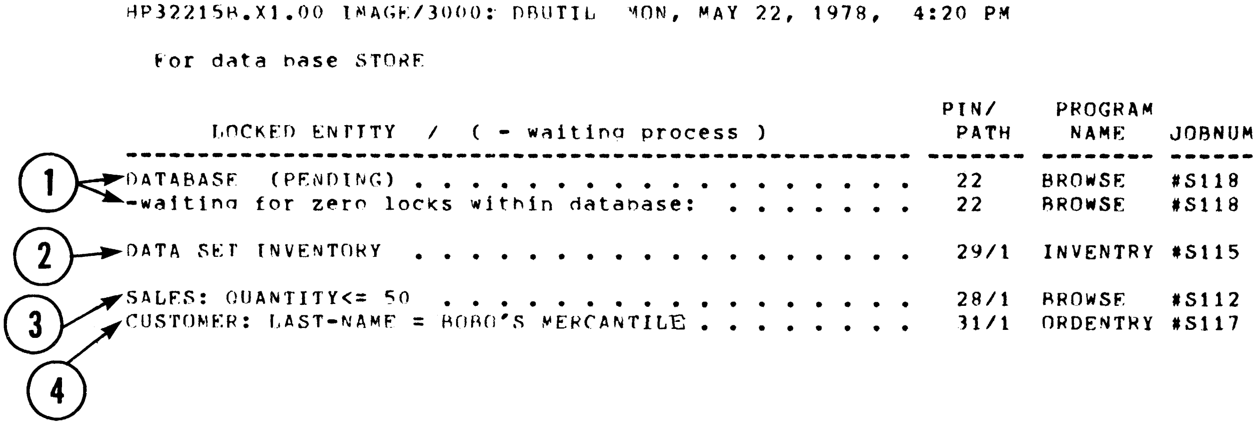
1. Indicates process 30 (program BROWSE executing in session 126) has the SALES data set locked through access path 1.

2. Shows a queue of processes waiting for the SALES data set to unlock. For example, in the first line, process 17 (program INVENTRY executing in session 128) is waiting. Since it is listed first in the queue, it will be the next process to resume execution after the SALES data set is unlocked. It may be waiting to place a lock on the data set or entries in the set.

3. Indicates process 30 (program BROWSE, session 126, access path 1) has the CUSTOMER data set locked. No processes are waiting for the lock to be released.

*This message is preceded by a hyphen so that it can be identified on terminals or listings from a line printer without lower case.

Here is another example of a locking list that might appear when the SHOW LOCKS command is entered.



```
HP32215B.X1.00 IMAGE/3000: DBUTIL  MON, MAY 22, 1978,  4:20 PM

  For data base STORE

                                                           PIN/    PROGRAM
        LOCKED ENTITY  /  ( - waiting process )            PATH     NAME    JOBNUM
---------------------------------------------------------  ------- -------- ------
DATABASE  (PENDING) . . . . . . . . . . . . . . . . . .    22     BROWSE   #S118
-waiting for zero locks within database:  . . . . . . .    22     BROWSE   #S118

DATA SET INVENTORY  . . . . . . . . . . . . . . . . . .    29/1   INVENTRY #S115

SALES: QUANTITY<= 50  . . . . . . . . . . . . . . . . .    28/1   BROWSE   #S112
CUSTOMER: LAST-NAME = BOBO'S MERCANTILE . . . . . . . .    31/1   ORDENTRY #S117
```

1. Indicates process 22 (program BROWSE, session 118) has obtained a lock on the data base and yet it cannot continue until existing locks held in the data base are released. In this example, the reason for the pending lock is listed on the line below.

2. Indicates process 29 (program INVENTRY, session 115, access path 1) has the INVENTORY data set locked.

3. Indicates that process 28 (program BROWSE, session 112, access path 1) has all entries in the SALES data set with QUANTITY less than or equal to 50 locked.

4. Indicates process 31 (program BROWSE, session 117, access path 1) has all entries in the CUSTOMER data set with LAST-NAME equal to BOBO'S MERCANTILE locked.

Note that all future requests for locks would be made to wait until process 22 releases its data base lock.

VERIFY

Reports whether a data-base-access file is activated or deactivated.

> The form of the VERIFY command is
>
> VER[IFY] *data-base-access file name*

For example,

```
VERIFY XXXDBA
           ↖
            data-base-access file name
```

where

data-base-access file name is the name of a data-base-access file.

EXAMPLE (Session Mode)

```
:RUN DBUTIL.PUB.SYS                 Initiate DBUTIL execution.
         .
         .
>>VERIFY XYZDBA                     Enter VERIFY command and data-base-
  Data-base-access file             access file name.
    XYZDBA is ACTIVATED
>>
```

DBSTORE

Stores the data base root file and all data set files to a tape or serial disc in a format compatible with back-up files created by the MPE :STORE and :SYSDUMP commands. DBSTORE differs from these commands in that it handles only IMAGE data bases.

Before copying the files, DBSTORE gains semi-exclusive access to the referenced data base; that is, DBSTORE determines that the only other data base activity consists of other users executing DBSTORE or application programs which open the data base in mode 6 or 8. If DBSTORE cannot gain semi-exclusive access, it terminates and prints the message DATA BASE IN USE.

You must be the data base creator or must provide the maintenance word to use DBSTORE.

OPERATION

```
① [:FILE DBSTORE; DEV=device]
②:RUN DBSTORE.PUB.SYS
           •
           •
③ WHICH DATA BASE? data base name [/maintenance word]
④ DATA BASE STORED

     END OF PROGRAM
```

where

device is the device class name of the device on which the data is to be stored.**

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.

1. Is an optional file equation which specifies the device class name for the device on which the data base is to be stored. The default is device class TAPE.

2. Initiates execution of the DBSTORE program which is in the PUB group and SYS account.

3. In session mode, DBSTORE prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.

4. After DBSTORE has copied the root file and all data set files, it prints a message to signal completion.

CONSOLE MESSAGES

After you have supplied the data base name and DBSTORE opens the output file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

**Serial disc packs must be initialized by the console operator using the VINIT subsystem. Refer to the *Console Operator's Guide*.

If more than one volume is required to store the data base, a request is displayed on the console for the next one. The next tape or disc must be mounted and the unit readied.** The volume which has been removed should be properly labeled with the data base name and volume number.

EXAMPLE (Job Mode)

```
              :EOJ               <------------ Terminate job.
           STORE/XYZED          <----------- Supply data base name and maintenance word.
       :RUN DBSTORE.PUB.SYS   <--------- Initiate DBSTORE program.
   :JOB MGR.ACCOUNTA          <-------- Initiate job.
```

After copying the STORE root file and all data sets, DBSTORE prints the following message on $STDLIST.

```
DATA BASE STORED
```

** The serial disc unit must be switched from "RUN" to "STOP" to "RUN" prior to entering a reply if a new volume has not been mounted. For example, this may occur if you restart the DBSTORE program.

DBRESTOR

Copies a data base from the backup volume(s) created by the DBSTORE program, or the MPE :STORE or :SYSDUMP commands, to disc.

If a catalogued disc file exists with the same name as the data base or the data base with two digits appended to it, DBRESTOR terminates and prints the message DUPLICATE FILE NAME. Therefore, it is recommended that you use the PURGE command of DBUTIL before executing DBRESTOR unless you know that the files do not exist in the account and group you are using for the data base.

To use DBRESTOR, you must either be the data base creator or supply the maintenance word. The DBRESTOR utility requires exclusive access to the data base.

OPERATION

```
① [:FILE DBRESTOR; DEV=device ]
② :RUN DBRESTOR.PUB.SYS
              •
              •
③ WHICH DATA BASE? data base name [/maintenance word]
④ DATA BASE RESTORED
     END OF PROGRAM
```

where

device — is the device class name of the device from which the data base is to be restored.

data base name — is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word — is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.

1. Is an optional file equation which specifies the device class name for the device from which the data base is to be restored. The default device class is TAPE.

2. Initiates execution of the DBRESTOR program which is in the PUB group and SYS account.

3. In session mode, DBRESTOR prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.

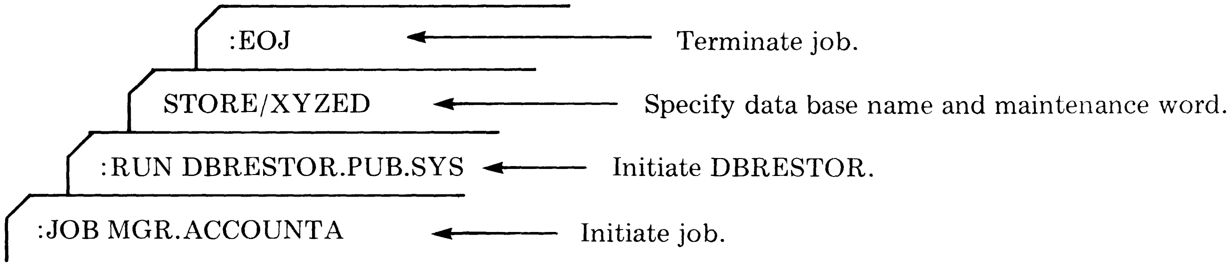
4. After DBRESTOR has created the root file and data set files and restored the data to these files, it prints a confirmation message.

CONSOLE MESSAGES

After you have supplied the data base name and DBRESTOR opens the input file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

If the data base is on more than one volume, another message is displayed on the system console. The operator must mount the next volume in the sequence and ready the unit. If the volume which is mounted is not the correct format, the operator is notified through a console message. If the correct volume is available, the current one should be removed and the correct one mounted. The operator must then enter a reply on the console.

EXAMPLE (Job Mode)



```
:EOJ                       Terminate job.
STORE/XYZED                Specify data base name and maintenance word.
:RUN DBRESTOR.PUB.SYS      Initiate DBRESTOR.
:JOB MGR.ACCOUNTA          Initiate job.
```

After creating the files and restoring the file contents, DBRESTOR prints the following message on $STDLIST:

```
DATA BASE RESTORED
```

DBUNLOAD

Copies the data entries from each data set to specially formatted tape or serial disc volumes.

The root file is not copied to the backup volume(s). Only data entries from non-empty records are copied. None of the pointer or structure information associated with the data entry is transferred.

The data sets are unloaded in the order that they were defined in the original schema. No data set names are recorded on the backup volume(s) ; entries are merely associated with the number of the data set from which they are read. DBUNLOAD calls the DBGET procedure to read each entry from each set of the data base using a *list* parameter of @; to read the complete entry. Values for data items appear in each entry in the same order as the items were mentioned in the data set definition in the schema.

DBUNLOAD requires exclusive access to the data base. If the data base is already open by any other process, DBUNLOAD prints the message: DATA BASE IN USE and prompts again for a data base name.

DBUNLOAD operates in either chained or serial mode. The mode is determined by the entry point specified with the :RUN command. The default entry, if none is specified, is CHAINED.

- In serial mode, DBUNLOAD copies the data entries serially in record number order. "Stand-alone" detail data sets, those which are not tied to any master data sets through specified search item paths, are always unloaded serially.

- In chained mode, DBUNLOAD copies all of the detail entries with the same primary path search item value to contiguous locations on the backup file. The ordering of the search item values from the primary path is based on the physical order of the matching value in the associated master data set. Figure 6-1 illustrates the method for unloading a data set in chained mode.

OPERATION

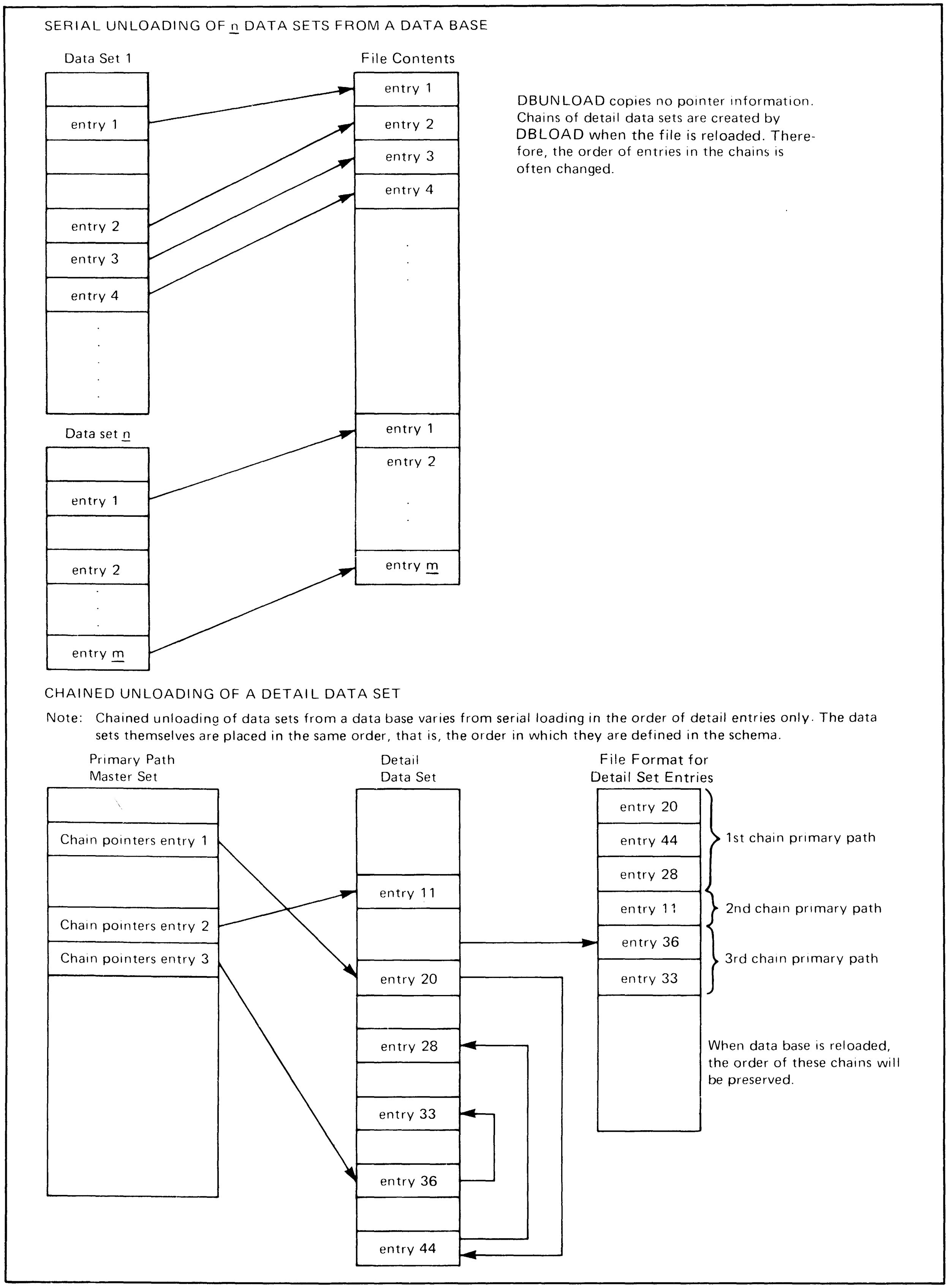
```
① [:FILE DBUNLOAD; DEV=device]  [,CHAINED]
② :RUN DBUNLOAD.PUB.SYS         [,SERIAL ]
              •
              •
③ WHICH DATA BASE? data base name [/maintenance word]
              •
              •
              •
④ DATA SET n: x ENTRIES
⑤ END OF VOLUME m, y, WRITE ERRORS RECOVERED
⑥ DATA BASE UNLOADED

       END OF PROGRAM
```

where

device is the device class name of the device to which the data entries are to be copied.**

data base name is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

**Serial disc packs must be initialized by the console operator with the VINIT subsystem. Refer to the *Console Operator's Guide.*



DBUNLOAD copies no pointer information. Chains of detail data sets are created by DBLOAD when the file is reloaded. Therefore, the order of entries in the chains is often changed.

Note: Chained unloading of data sets from a data base varies from serial loading in the order of detail entries only. The data sets themselves are placed in the same order, that is, the order in which they are defined in the schema.

When data base is reloaded, the order of these chains will be preserved.

Figure 6-1. DBUNLOAD File, Sequence of Entries

maintenance word is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.

n is the number of data sets in the data base.

x is the number of entries copied from the specified data set.

m is the number of the volume.

y is the number of write errors from which DBUNLOAD has successfully recovered.

1. Is an optional file equation which specifies the device class name for the device on which the data entries are to be copied. The default is device class TAPE.

2. Initiates execution of the DBUNLOAD program which is in the PUB group and SYS account.

3. In session mode, DBUNLOAD prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.

4. After each data set is copied, DBUNLOAD prints a message which includes the data set number and the number of entries copied.

   If the number of entries copied differs from the expected number of entries, DBUNLOAD sends the additional message:

   *****NUMBER OF ENTRIES EXPECTED *y*

   where *y* is the number of entries the data set contains according to the root file.

5. When the end of a volume is encountered, DBUNLOAD prints this message:

   END OF VOLUME *a*, *b*, WRITE ERRORS RECOVERED

   where *a* is the number of the volume and *b* is the number of write errors from which DBUNLOAD successfully recovered. DBUNLOAD also instructs the operator to save the current volume and mount a new one by printing the following two messages on the system console:

   SAVE VOLUME ON LOGICAL DEVICE *n* AS *XXXXX y*
   MOUNT NEXT VOLUME ON LOGICAL DEVICE *n*

   where *n* is the logical device number of the tape or disc drive, *XXXX* is the data base name, and *y* is the volume number.

6. After the data sets have been successfully copied, DBUNLOAD issues a completion message.

CONSOLE MESSAGES

After you have supplied the data base name and DBUNLOAD opens the output file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

USING CONTROL Y

When executing DBUNLOAD in session mode, you can press Control Y to request the approximate number of entries in the current data set which have already been written. DBUNLOAD then prints:

```
<CONTROL Y> DATA SET n: x ENTRIES HAVE BEEN PROCESSED
```

on $STDLIST.

WRITING ERRORS

If an unrecoverable write error occurs, DBUNLOAD prints the message:

```
UNRECOVERABLE WRITE ERROR, RESTARTING AT BEGINNING OF VOLUME
```

and attempts to recover by starting the current volume again. It also sends this message to the system operator:

```
WRITE PROBLEMS TRY ANOTHER VOLUME ON LOGICAL DEVICE n
```

where *n* is the logical device number of the unit.

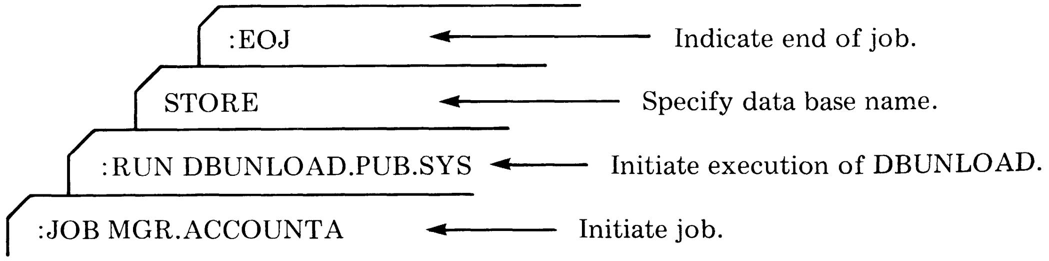
If an excessive number of non-fatal write errors occur, DBUNLOAD again attempts to recover from the beginning of the volume after printing the following message on the $STDLIST.

```
EXCESSIVE WRITE ERROR RECOVERIES, RESTARTING AT BEGINNING OF VOLUME
```

and sends the same message to the system operator as described for unrecoverable errors above.

EXAMPLE (Job Mode)

The job input appears as follows:



```
:EOJ                      ← Indicate end of job.
STORE                     ← Specify data base name.
:RUN DBUNLOAD.PUB.SYS     ← Initiate execution of DBUNLOAD.
:JOB MGR.ACCOUNTA         ← Initiate job.
```

Since the user in this example is the data base creator, a maintenance word is not provided. The DBUNLOAD program is executed in CHAINed mode by default because no entry point is specified.

As the job executes, the following information is printed on the $STDLIST:

```
DATA SET 1:        9 ENTRIES
DATA SET 2:       50 ENTRIES
DATA SET 3:       24 ENTRIES
DATA SET 4:       12 ENTRIES
DATA SET 5:        5 ENTRIES
DATA SET 6:       10 ENTRIES
END OF VOLUME 1, 0 WRITE ERRORS RECOVERED
DATA BASE UNLOADED
END OF PROGRAM
```

DBLOAD

Loads data entries from the backup volume(s) created by the DBUNLOAD program into data sets of the data base.

The volume(s) must have been produced by the DBUNLOAD program, and the data base name on the volume must be exactly the same as the data base name, or root file name, in the current session or job's group and account. To reload the identical data into the data base, the DBUTIL ERASE command must be used prior to DBLOAD unless the data base has been purged and re-created. Executing DBUTIL in this way reinitializes the data sets to an empty state while keeping the root file and data sets as catalogued MPE files on the disc.

DBLOAD reads each entry from the backup volume and puts it into the same numbered data set from which it was read by DBUNLOAD. If a data set in the receiving data base is an automatic master, no entries are directly put into it by DBLOAD, even though there are entries on the volume associated with the data set's number. Automatic master entries are created as needed in the normal fashion when entries are put into the detail data sets related to the automatic master.

DBLOAD calls the DBPUT procedure to put the entries read from the backup volume into the appropriate data sets. In every case, the DBPUT *dset* parameter is a data set number and the *list* parameter is @;. Prior to calling DBPUT, DBLOAD moves each entry from the backup volume into a buffer. The length of the entry is determined by the definition of entries in the target data set. When DBLOAD is calling DBPUT, this length is less than, equal to, or greater than the length of an entry on the backup volume. If the data set entry is larger than the backup entry, the data is left-justified and is padded out to the maximum entry length with binary zeroes. If the data entry is smaller than the backup entry, the backup volume record is truncated on the right and the truncated data is lost.

The location of master set entries is based on their search item value. The detail data set entries are put into consecutive data set records with the appropriate new chain pointer information.

DBLOAD requires exclusive access to the data base. If the data base is already open to any other process, DBLOAD terminates and prints the message: DATA BASE IN USE.

OPERATION

```
① [:FILE DBLOAD; DEV=device]

② :RUN DBLOAD.PUB.SYS
        •
        •
③ WHICH DATA BASE? data base name [/maintenance word]

④ DATA SET 1: x ENTRIES
        •
        •
        •
⑤ END OF VOLUME m,y READ ERRORS RECOVERED

⑥ DATA BASE LOADED

   END OF PROGRAM
```

where

device — is the device class name of the device from which the data entries are to be loaded.

data base name — is the name of an IMAGE data base root file catalogued in the current session or job's account and log on group.

maintenance word — is the maintenance word defined by the data base creator. This word must be supplied by anyone other than the data base creator.

n — is the number of the last data set loaded from the backup volume.

x — is the number of entries loaded into the specified data set. *x* is zero if the data set is an automatic master. Note: this number may not represent the total number of records in the data set if entries existed prior to DBLOAD execution.

m — is the volume number.

y — is the number of read errors from which DBLOAD recovered.

1. Is an optional file equation which specifies the device class name for the device from which the data entries are to be loaded. The default is device class TAPE.

2. Initiates execution of the DBLOAD program which is in the PUB group and SYS account.

3. In session mode, DBLOAD prompts for the data base name and maintenance word. In job mode, the data base name and maintenance word, if any, must be in the record immediately following the :RUN command.

4. After each data set is copied, DBLOAD prints a message on the *listfile* device which includes the data set number and the number of entries copied.

5. When the end of a volume is encountered, DBLOAD prints this message. DBLOAD also instructs the operator to mount a new one with the following message on the system console:

   MOUNT DBLOAD VOLUME *XXXXy* ON LOGICAL DEVICE *n*

   where *n* is the logical device number of the unit, *XXXX* is the data base name, and *y* is the volume number.

   If the operator mounts the wrong volume, DBLOAD informs the operator with the message:

   WRONG VOLUME MOUNTED ON LOGICAL DEVICE *n*

   where *n* is the logical device number.

   DBLOAD then terminates and you must begin loading the data base again. This may require running DBUTIL,ERASE again if you have already loaded any volumes.

6. After the data entries have been successfully loaded, DBLOAD prints a completion message.

CONSOLE MESSAGES

After you have supplied the data base name and DBLOAD opens the input file, a message is displayed on the system console. A tape or serial disc must be mounted on the appropriate unit and identified through an operator reply. Refer to the *Console Operator's Guide* for instructions about console interaction.

USING CONTROL Y

When executing DBLOAD in session mode, you can press Control Y to request the approximate number of entries in the current data set that have already been copied. DBLOAD prints:

<CONTROL Y> DATA SET *n:x* ENTRIES HAVE BEEN PROCESSED

on $STDLIST.

EXAMPLE (Session Mode)

```
 :RUN DBLOAD.PUB.SYS

WHICH DATA BASE? STORE/XYZED
DATA SET 1:  19 ENTRIES
DATA SET 2:   0 ENTRIES
DATA SET 3:  25 ENTRIES
DATA SET 4:  12 ENTRIES
DATA SET 5:  32 ENTRIES
DATA SET 6: 258 ENTRIES
END OF VOLUME 1, 0 READ ERRORS RECOVERED
DATA BASE LOADED

  END OF PROGRAM
```

Initiate execution of DBLOAD.

Supply data base name and maintenance word. DBLOAD indicates number of entries copied. Data set 2 is an automatic master so 0 entries are copied; the entries are created as related detail entries are copied to the data base.

One volume was copied with no read errors.

NOTE: For optimum performance, DBLOAD uses a special "output deferred" mode of operation when it adds entries to a data base. In this mode, data and structural information may not be written back to disc each time DBPUT returns to the DBLOAD program. As a result, the data base is not considered to be logically or structurally complete on disc until the DBLOAD is complete. During DBLOAD the data base being loaded is marked "bad", and only at the completion of a DBLOAD run is the data base marked "good" again.

If, during a load, an MPE or hardware crash occurs, the data base is definitely not structurally intact, and it returns its "bad" flag. After the system is brought back up, IMAGE does not allow the data base to be opened for normal access. If you get a "bad data base" error in such a situation, you should erase the data base with DBUTIL and then perform the load again. Alternatively, you may purge the data base with DBUTIL and then restore it from a backup copy. A "bad" data base may also be stored, restored, or unloaded (in serial mode only).

INTERNAL STRUCTURES AND TECHNIQUES

SECTION VII

In addition to the data elements discussed in Section II, data bases consist of a number of structure elements. IMAGE creates and uses these, along with various internal techniques, to provide rapid and efficient access to the data base content. This section describes these structures and techniques to give you a complete understanding of the way IMAGE works. A summary of design considerations is included at the end of this section.

STRUCTURE ELEMENTS OF DATA SETS

The following internal structures are used by IMAGE to manage the information in data sets.

POINTERS

IMAGE uses pointers to link one data set record to another. A pointer is a two word entity (double-word) containing the relative address of a record within a data set.

DATA CHAINS

A data chain is a set of detail data set entries that are bidirectionally linked together by pairs of pointers. All entries having a common search item value are placed in the same chain. Each chain has a first and a last member. The pointer pairs constitute backward and forward pointers to the entry's predecessor and successor within the chain. The first member of a chain contains a zero backward pointer and the last member of a chain contains a zero forward pointer. A single chain may consist of at most 65535 entries.

MEDIA RECORDS

IMAGE transfers information to and from a storage location on disc in the form of a media record. A media record consists of both an entry and its pointers or a null record if no data entry is present.

MEDIA RECORDS OF DETAIL DATA SETS

For each detail entry, the media record consists of the entry itself preceded by all of its related data chain pointer pairs. The number of pointer pairs corresponds to the number of paths specified for the data set within the schema. Figure 7-1 illustrates a media record for a detail data set defined with two paths. The first set of pointers corresponds to the first path defined in the set part of the schema and the second set corresponds to the second path.

backward pointer path 1	forward pointer path 1	backward pointer path 2	forward pointer path 2	entry . . . .

Figure 7-1. Media Record for Detail Entry

CHAIN HEADS

IMAGE locates the first or last member of a chain within a detail data set by using a *chain head*. The chain head for a particular chain is stored with the entry in the corresponding master data set whose search item value is the same as the detail search item value defining the chain. Each chain head is five words long. The first word is an integer count of the number of member entries in the referenced chain. The count is zero if the chain is empty. The remaining four words contain two doubleword pointers. One points to the last chain entry, the other to the first chain entry. If the count is zero, these pointers are both zero. If the count is 1, these pointers have the same value.

PRIMARY ENTRIES

Selection of record addresses for master entries begins with a calculated address determined by an algorithm applied to the value of each entry's search item. The algorithm is described later in this section. Each such calculated address is known as a *primary address* and each entry residing at its primary address is called a *primary entry.*

SECONDARY ENTRIES

Occasionally a new entry with a unique search item value is assigned the same primary address as an existing primary entry since the search item values of both entries generate the same calculated address. When this occurs, the entries are called *synonyms.* IMAGE assigns the new entry a *secondary address* obtained from unused records in the vicinity of the primary entry. All entries residing at secondary addresses are called secondary entries.

SYNONYM CHAINS

A primary entry may have multiple synonyms. A *synonym chain* consists of the primary entry and all of its synonyms. Each synonym chain is maintained by a five-word chain head in the media record of the primary entry and five-word links in the media records of the secondary entries. Note that a master data set entry may contain both a synonym chain head and multiple detail chain heads. These are two distinct types of chain heads.

If no secondary entries are present the synonym chain count is 1 and the pointers to the first and last secondary entries are zeroes. If N secondaries are present, the chain count is N+1 and the pointers reference the first and last secondary entries.

The first word of the five-word link in the media record of each secondary entry is always zero. The remaining four words consist of two doubleword pointers bi-directionally linking the secondary entries of the synonym chain to each other. As with detail chains, the first member of the synonym chain contains a zero backward pointer and the last member of the chain contains a zero forward pointer.

MEDIA RECORDS OF MASTER DATA SETS

Media records of master data entries are composed of the following:

- A five-word field serving as a synonym chain head for primary entries or a synonym chain link for secondary entries.
- A 5 times *n* word field in which the chain heads of all related detail chains are maintained. *n* is the number of relationships defined for the master data set. There may be between 0 and 16 relationships.
- The entry itself.

Figure 7-2 illustrates the media record for a primary entry of a master data set with two paths defined.

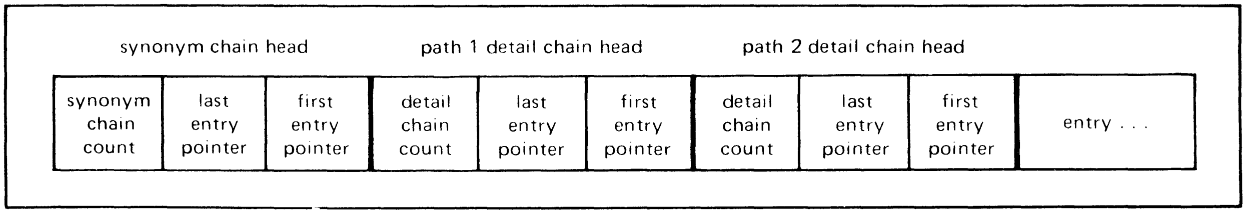


Figure 7-2. Media Record for Primary Entry

Figure 7-3 illustrates a media record for a secondary entry of a master data set with two paths defined.

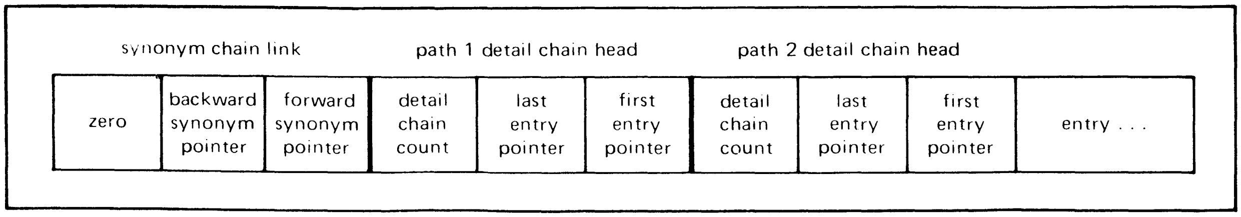


Figure 7-3. Media Record for Secondary Entry

When more than one detail chain head is present, they are physically ordered left-to-right in the order that the associated paths are specified in the schema.

BLOCKS AND BIT MAPS

Each group of media records involved in a single disc transfer is a *block*. The first word or words of each block contain a bit map employed by IMAGE to indicate whether a particular media record of the block contains a data set entry or is empty. There is one bit for each record in the block. The bits occur in the bit map in the same order that the records occur in the block. The bit map occupies as many integral words as are required to contain one bit for each record in the block. If a bit is zero, the corresponding record is empty. If a bit is one, the record contains a data entry preceded by the associated structure information.

The format of a block is illustrated in figure 7-4. The sample block contains four records and the third record contains no entry.

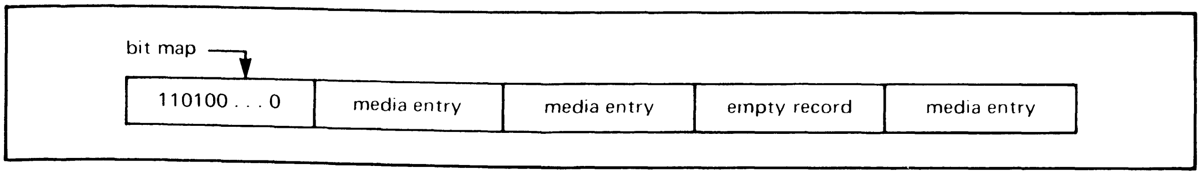


Figure 7-4. Block with Blocking Factor of Four

RUN-TIME IMAGE CONTROL BLOCKS

As mentioned in Section IV, IMAGE uses dynamically-constructed control blocks resident in privileged extra data segments to provide and control user access to a data base through the IMAGE procedures. The contents of these control blocks are maintained by IMAGE, and it is not necessary to know the details in order to use the IMAGE procedures. However, the following descriptions are provided for those who prefer to understand the control blocks and their functions.

LOCAL DATA BASE ACCESS

Two structures are involved in local data base access:

- The global *Data Base Control Block* (DBCB), and
- the *User Control Block* (ULCB).

For any open IMAGE data base, there is exactly one DBCB, regardless of the number of users. The DBCB is created by IMAGE when the first user opens a data base. It is shared by all concurrent users and is released when the last user closes the data base.

The DBCB is derived from the root file and contains all information (static and dynamic) about the data base which is common to all users. In addition, the DBCB contains a group of input/output (I/O) buffers and a communications area used by the IMAGE procedures. The I/O buffers vary in number with the number of concurrent users and can be controlled by the data base administrator. (Refer to the DBUTIL program description in Section VI.) These buffers are shared by all data sets and by all concurrent users; each is as large as the largest block of the data base.

One ULCB exists for each user of each open data base. A ULCB is created each time a user opens a data base and is released when the user closes the data base. A ULCB is typically less than ten percent the size of the corresponding DBCB and contains information (static and dynamic) which is unique to a particular user's access to a data base. Examples of this type of information are the user's access mode, security data, current position in each data set, and current list for each data set.

When accessing a local data base, the IMAGE procedures usually make use of and modify information in both the global DBCB and the specific user's ULCB for this data base. Specifically, during the processing of a single IMAGE call, the necessary information is moved from the appropriate ULCB into a reserved area in the DBCB where it is used. Information that has changed is moved back to the ULCB at the conclusion of the call.

REMOTE DATA BASE ACCESS

IMAGE provides the capability of accessing a data base on a remote HP 3000 from a user program running on the local HP 3000, as described in Section VIII. This capability is provided in conjunction with DS/3000 and is accomplished by transmitting IMAGE data base access requests (DBGET, DBPUT, and so forth) to the remote computer where they are executed and the results returned to the local calling program. The control block structures used by IMAGE on the remote computer which contains the data base are those described in the preceding paragraphs.

On the local computer which is running the user application program, a structure called the Remote Data Base Control Block (RDBCB) is constructed and used by IMAGE. One RDBCB exists for each user accessing a remote IMAGE data base (each access path to a remote data base). The RDBCB is created when the user opens the data base and is released when the user closes the data base. It resides in a privileged extra data segment associated with the user application process on the local computer. The RDBCB contains data base, set, and item information plus the work areas necessary to process IMAGE procedure calls and pass them efficiently over the DS/3000 line for execution on the remote computer. Returned data and status information is also processed in the RDBCB and is transferred to the appropriate user stack areas before IMAGE returns to the local calling program.

CONTROL BLOCK SIZES

It is impossible to predict the exact length of the control blocks used by IMAGE to manage user access to data bases. However, table 7-1 contains formulas that provide a way of approximating their sizes.

The exact length of the ULCB and the exact current length of the DBCB are returned in the *status* array by DBOPEN. (These lengths do not include the few words of overhead used by MPE.)

Table 7-1. Formulas for Approximating Control Block Sizes

CONTROL BLOCK	APPROXIMATE LENGTH (IN WORDS)
DBCB	r + i + s + (n (b + 9)) + (12u) + t + 150
ULCB	i + (15s) + (f/2) + 10
RDBCB	(12i) + (13s) + 160 + max(256,129+e)
in which	
r	is the ROOT LENGTH as reported by the Schema Processor.
b	is the BUFFER LENGTH as reported by the Schema Processor.
t	is the initial TRAILER LENGTH as reported by the Schema Processor. (The trailer area may expand.)
n	is the number of I/O buffers in the DBCB. This number normally varies with the number of concurrent users of a data base. DBUTIL can be used to display and change the values which IMAGE will use.
i	is the number of data items in the data base.
s	is the number of data sets in the data base.
f	is the number of "fields" in the data base. Count each data item once for each data set in which it appears in the schema.
u	is the number of concurrent users of the data base.
e	is the largest entry length of any data set in the data base. Derive this from the ENTR LGTH column of the summary table printed by the Schema Processor.

INTERNAL TECHNIQUES

Although it is not necessary to know the following techniques to use IMAGE, an understanding of them may help you design a more efficient data base.

PRIMARY ADDRESS CALCULATION

IMAGE employs two distinct methods of calculating primary addresses in master data sets. The first method applies to master data sets with search items of type I, J, K, or R. The low order (rightmost) 31 bits of the search item value, or the 16 bits of a one-word search item value, are used to form a 32-bit doubleword value. This doubleword value is then decremented by one, reduced modulo the data set capacity and incremented by one to form a primary address. For example, if an integer search item has a value of 529 and the data set capacity is 200, the primary address is 129. The calculation is as follows:

$$\begin{array}{rr l} 529 - 1 = & 528 & \\ & \underline{-400} & \quad (200 \times 2) \\ & 128 & \\ & \underline{+1} & \\ & 129 & \end{array}$$

The second method of primary address calculation applies to master data sets with search items of type U, X, Z, or P. In this case, the entire search item value regardless of its length is used to obtain a positive doubleword value. This value is reduced modulo the data set capacity and then incremented by one to form a primary address. The algorithm used to obtain the doubleword intermediate value attempts to approximate a uniform distribution of primary addresses in the master data set, regardless of the bias of the master data set search item values.

The intent of the two primary address algorithms is to spread master entries as uniformly as possible throughout the record space of the data file. This uniform spread reduces the number of synonyms occurring in the master data set.

NOTE

Generally, a master data set with a capacity equal to a prime number or to the product of two or three primes yields fewer synonyms than master data sets with capacities consisting of many prime factors.

MIGRATING SECONDARIES

In some cases, secondary entries of master data sets are automatically moved to storage locations other than the one originally assigned. This most often occurs when a new master data entry is assigned a primary address occupied by a secondary entry. By definition, the secondary entry is a synonym to some other primary entry resident at their common primary address. Thus, the new entry represents the beginning of a new synonym chain. To accommodate this new chain the secondary entry is moved to an alternate secondary address and the new entry is added to the data set as a new primary entry. This move and the necessary linkage and chain head maintenance is done automatically.

A move can also occur when the primary entry of a synonym chain having one or more secondary entries is deleted. Since retrieval of each entry occurs through a synonym chain, each synonym chain must have a primary entry. To maintain the integrity of a synonym chain, IMAGE always moves the first secondary entry to the primary address of the deleted primary entry. The former first secondary entry is now the primary entry for the chain and the record formerly containing the secondary entry is now empty.

SPACE ALLOCATION FOR MASTER DATA SETS

Space allocation for each master data set is controlled by a doubleword free space counter resident in the user label of the data set, and by all the bit maps, one in each block of the data set.

When a new entry is added, IMAGE decrements the free space counter and sets the bit corresponding to the newly assigned record address to a 1. If the bit is a zero before the record is added, the assigned record address is the primary address. If the bit is a one before the record is added, it indicates that a primary entry already exists. In this case, a secondary address is determined by a cyclical search of the bit maps for a 0 indicating an unused record and this address is assigned to the entry being added.

SPACE ALLOCATION FOR DETAIL DATA SETS

Space allocation for each detail data set is controlled by a doubleword free space counter, a doubleword *end-of-file pointer* and a doubleword pointer to a *delete chain.* The end-of-file pointer contains the record address of the highest-numbered entry which has existed so far in the data set. The delete chain pointer locates the record from which an entry was most recently deleted. When each detail data set is first created, the end-of-file pointer and delete chain pointer are both zero.

When a new entry is added to a detail data set, IMAGE assigns to it the record address referenced by the delete chain pointer, unless the pointer is zero. If the chain pointer is zero, the end-of-file pointer is incremented and then used as the assigned record address. The free space counter is decremented in either case.

When an existing entry is deleted, its media record is zeroed, the first two words are replaced with the current delete chain pointer, and the block is written to disc. The delete chain pointer is set to the address of the newly deleted entry and the free space counter is incremented.

The delete chain is, in effect, a push down stack of reusable media record space. Reusable space is always allocated in preference to the unused space represented by the record addresses beyond the end-of-file pointer.

Addition and deletion of data entries also requires data chain maintenance and the turning on or turning off the corresponding bit of the appropriate bit map. Both of these are necessary for retrieval integrity but neither play a role in space allocation for detail data sets.

LOCKING INTERNALS

Within the DBCB is a "lock area" which is initially 128 words long and may be expanded to 4096 words in length subject to system availability of the necessary resources. It contains a fixed area of 64 words. The remaining area is managed dynamically and provides space for the following entries:

- Accessor Entries — Each of these is 8 words long. One is created for each successful call to DBOPEN (each access path). Although located in the lock area, this is the master entry with which IMAGE controls access to the data base. It disappears when DBCLOSE is called for the access path and the space is reused.

- Set Entries — Each of these is 16 words long. One is created for every data set that is specified in a lock request. Therefore, the maximum number of set entries is equal to the number of data sets in the data base. These are never deleted.

- Descriptor Entries — These entries contain the internal form of the lock descriptors specified in locking mode 5 or 6. Each entry is 8 + V words long rounded up to the next multiple of 8. V is the number of words required to hold the value of the data item used for locking. These entries disappear when the locks are released (when DBUNLOCK is called) and the space is reused.

USING A REMOTE DATA BASE

SECTION VIII

If you want to access a data base that resides on one HP 3000 computer system while operating a session on another HP 3000 computer system, you may do so provided both systems are configured with Distributed Systems (DS/3000) capability. If you are not familiar with DS/3000 you should read the *DS/3000 Reference Manual* before you begin accessing a remote data base.

DS/3000 refers to the computer to which your terminal is directly connected as the local HP 3000 and the computer with which you establish a communications link as the remote HP 3000. The session that you initiate on the local HP 3000 is a local session and a session on a remote HP 3000 is a remote session.

You may use a data base on a remote HP 3000 either from a program that is running on the remote system or from a program running on your local HP 3000. There are various ways to open a communications line and initiate a remote session. For example, you can establish a communications link and remote session and then run a remote program accessing a data base on the remote machine as illustrated in figure 8-1.

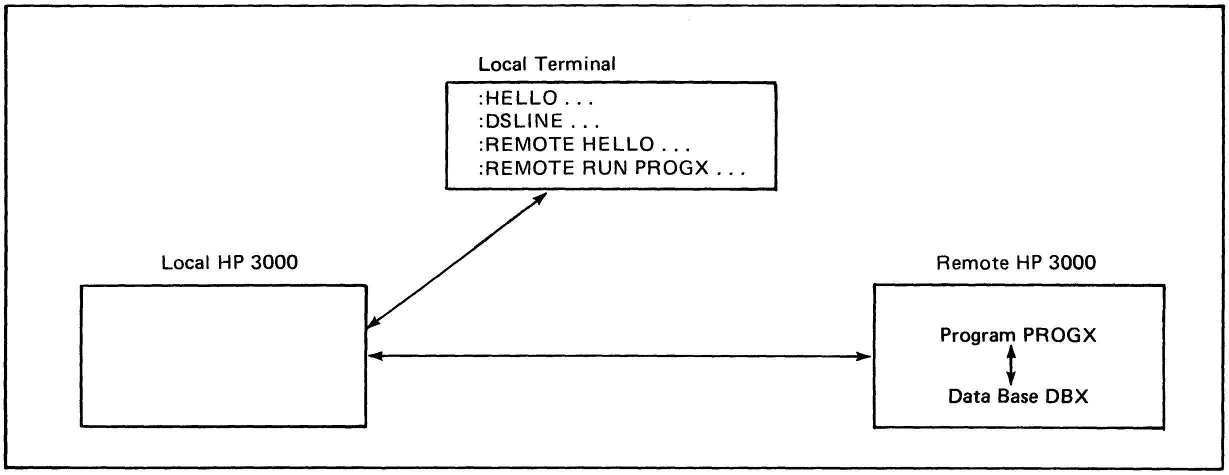


Figure 8-1. Using a Remote Program

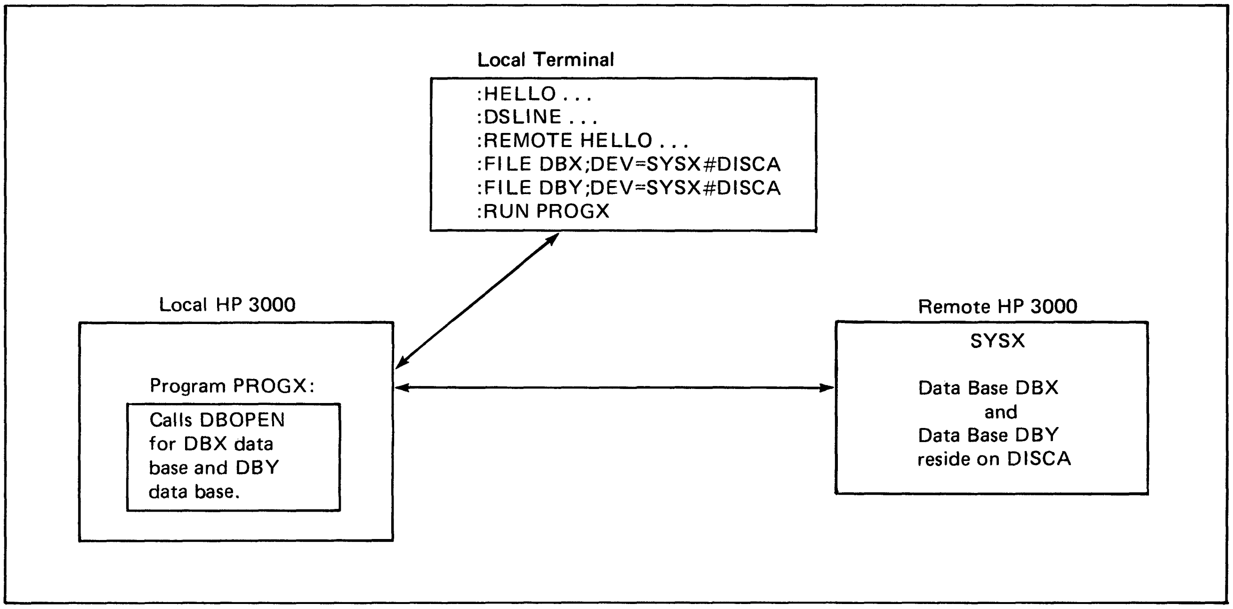
Refer to Sections I through III of the *DS/3000 Reference Manual* for a detailed description of this method.

ACCESS THROUGH A LOCAL APPLICATION PROGRAM

If you want to access a remote data base using a local application program, there are three methods you may choose from. In all cases, a local program accesses a remote data base and the data is passed across the communication line.

METHOD 1: ESTABLISHING COMMUNICATIONS LINK AND REMOTE SESSION INTERACTIVELY

To use the first method, you interactively establish a communications link and a remote session and enter a FILE equation for each remote data base. The FILE equation specifies which data base is to be accessed on which remote system and device. A local application program can now access a remotely located data base, as shown in figure 8-2.



```
:HELLO . . .
:DSLINE . . .
:REMOTE HELLO . . .
:FILE DBX;DEV=SYSX#DISCA
:FILE DBY;DEV=SYSX#DISCA
:RUN PROGX
```

Local HP 3000
Program PROGX:
Calls DBOPEN for DBX data base and DBY data base.
Remote HP 3000
SYSX
Data Base DBX and Data Base DBY reside on DISCA

Figure 8-2. Using Method 1

For details about using this method refer to the *DS/3000 Reference Manual.*

METHOD 2: USING THE COMMAND INTRINSIC

The second method is very similar to the first, but you use the MPE COMMAND intrinsic within your application program to establish the communications link, remote session and remote data base access. In order to use this method in an application program which is coded in COBOL, RPG, or BASIC you must write a procedure in SPL or FORTRAN and call the procedure.

To use this method you must issue a REMOTE HELLO command (either with the DSLINE parameter or issue the DSLINE as a separate command) and a FILE equation by calling the COMMAND intrinsic for each of these commands. Use of the COMMAND intrinsic is explained in the *MPE Intrinsics Reference Manual*, and information about accessing remote files is given in the *DS/3000 Reference Manual.* Figure 8-3 contains a diagram of Method 2.

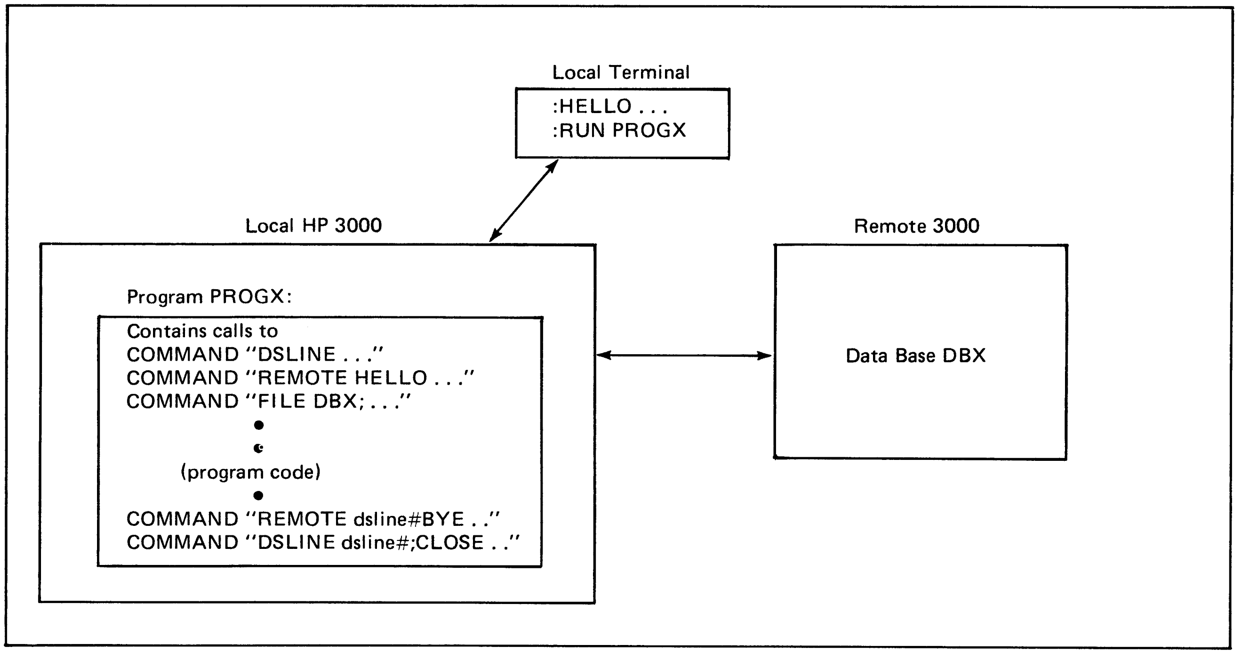


Figure 8-3. Using Method 2

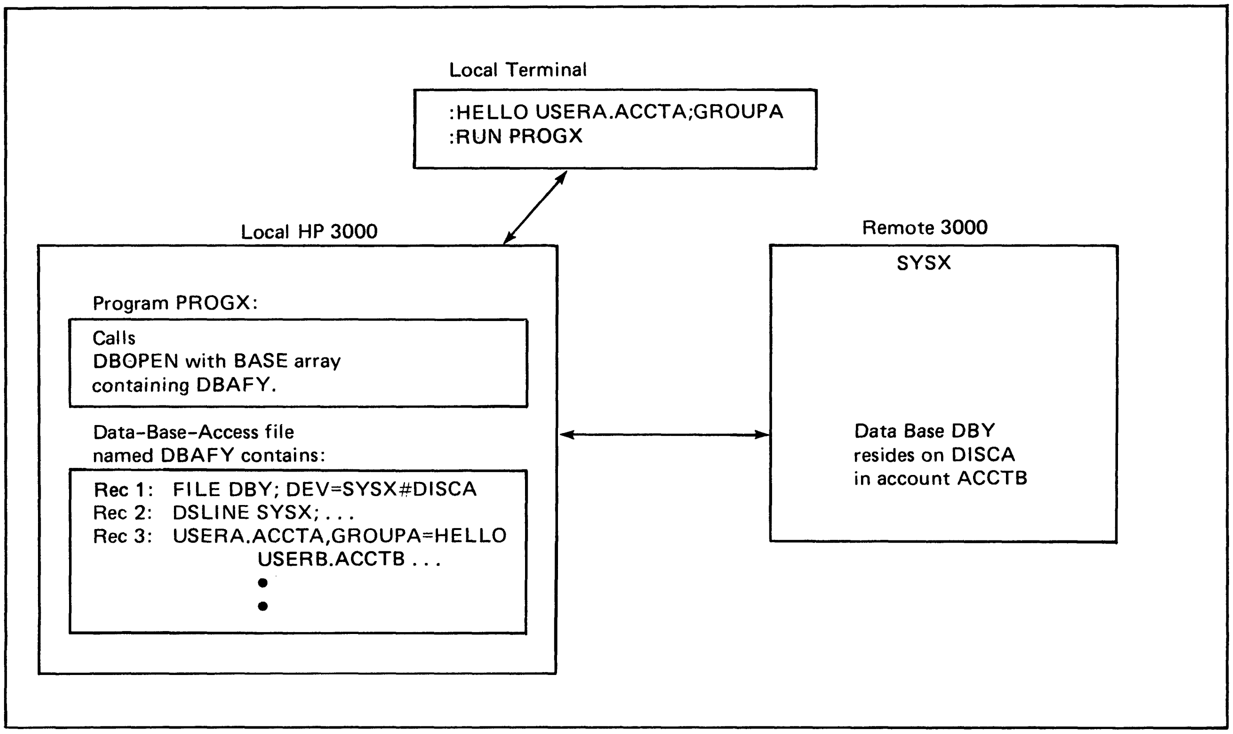
If you want to access more than one remotely located data base with an application program, you must enter one FILE equation for each remote data base. It is important to remember that a REMOTE HELLO to the same remote computer should not be repeated within a process since the second request for a remote session would log off the first one.

NOTE: When the application program calls the DBCLOSE procedure or is ready to terminate execution, it must programmatically issue a REMOTE BYE and DSLINE;CLOSE on the DSLINE specified with the foregoing COMMAND intrinsic.

If you use this method, any change in the data base name, account or password information requires modification of the application program.

METHOD 3: USING A DATA-BASE-ACCESS FILE

The third method involves creating a special file which we shall call the data-base-access file (DBA file). This file provides IMAGE with the necessary information to establish a communications link and a remote session. It also specifies the remote data base or data-base-access file name so that the necessary IMAGE intrinsics can be executed on the remote computer.



```
:HELLO USERA.ACCTA;GROUPA
:RUN PROGX
```

```
Rec 1:  FILE DBY; DEV=SYSX#DISCA
Rec 2:  DSLINE SYSX;...
Rec 3:  USERA.ACCTA,GROUPA=HELLO
                USERB.ACCTB ...
                •
                •
```

Figure 8-4. Using Method 3

NOTE

It should be noted that with Method 3, which uses the data-base-access file, only one data base can be accessed using each data-base-access file per DSLINE. For example, if two computers are linked through two DSLINEs you can open one data base on each line. A second REMOTE HELLO on one DSLINE terminates the previous REMOTE HELLO. For multiple remote data base access, Method 1 or Method 2 is recommended. If the data-base-access file is used, an automatic REMOTE BYE and DSLINE;CLOSE is issued on the DSLINE specified in the data-base-access file when the application program terminates execution.

By using this approach, the data base administrator can set up a user-table which provides more control over the data base users, and thus, enhances data base security. To create the data-base-access file, you use the Editor (EDIT/3000). The content of this file should be created in the format shown below.

FORMAT

Record 1	FILE *dbname1*[=*dbname2*] ; DEV = *dsdevice#device*
Record 2	DSLINE *dsdevice* [;LINEBUF=*buffer-size*] [;LOCID=*local-id-sequence*] [;REMID=*remote-id-sequence*] [;PHNUM=*telephone-number*] [;EXCLUSIVE] [;QUIET]
Record 3	*lusername.lacctname* [,*lgroupname*]=HELLO *rusername* [/*rupasw*]. *racctname*[/*rapasw*] [,*rgroupname*] [/*rgpasw*] [;TIME=*cpusecs*] [;PRI=*priority*] [;HIPRI ;INPRI=*inputpriority*]
Records 4 through n	Same format as record 3. Specifies other "user.account,group" identification.

PARAMETERS

dbname1

is the name of the data base or the data-base-access file on the remote system you want to access, or is the formal file designator used in the program if *dbname2* is specified. (Required parameter)

dbname2

is the name of the data base or the data-base-access file on the remote system you want to access. (Optional parameter)

dsdevice

is the device class name or logical device number assigned to the DS/3000 communications driver IODSO during system configuration. (Required parameter)

device

is the device class name or logical device number of the remote device on which the data base resides. (Required parameter)

buffer-size

is a decimal integer specifying the size (in words) of the DS/3000 line buffer to be used in conjunction with the communications line. The integer must be within the range 304 < *buffer-size* < 4096. The default value is the buffer size entered in response to the PREFERRED BUFFER SIZE prompt during system configuration. (Optional parameter)

local-id-sequence

is a string of ASCII characters contained within quotation marks or a string of octal numbers separated by commas and contained within parentheses. If you wish to use a quotation mark within an ASCII string, use two successive quotation marks. In the case of an octal sequence, each octal number represents one byte and must be within the range 0-377. The maximum number of ASCII characters or octal numbers allowed in the string is 16.

The supplied string of ASCII characters or octal numbers define the ID sequence that will be sent from your HP 3000 to the remote HP 3000 when you attempt to establish the telephone connection. If the remote HP 3000 does not recognize the supplied ID sequence as a valid one, the telephone connection is terminated. The default value is the ASCII or octal string entered in response to the LOCAL ID SEQUENCE prompt during system configuration. (Optional parameter)

remote-id-sequence Same format as local-id-sequence.

The supplied string of ASCII characters or octal numbers define those remote HP 3000 ID sequences that will be considered valid when you attempt to establish the telephone connection. If the remote HP 3000 does not send a valid ID sequence, the telephone connection is terminated. The default set of remote ID sequences consists of the ASCII and octal strings entered in response to the REMOTE ID SEQUENCE prompt during system configuration. (Optional parameter)

telephone-number is a telephone number consisting of digits and dashes. The maximum length permitted (including both digits and dashes) is 20 characters. If YES was entered in response to the DIAL FACILITY prompt during system configuration, this telephone number will be displayed at the operator's console of your HP 3000 and the operator will then establish the telephone connection by dialing that number at the MODEM. The default telephone number is the first one entered in response to the PHONE NUMBER prompt during system configuration.(Optional parameter)

EXCLUSIVE This parameter, if present, specifies that you want exclusive use of the particular communications line. If the specified SSLC is already open and you have specified the exclusive option, DS/3000 will deny you access to the line (you cannot open it). Opening an EXCLUSIVE line requires the user to have CS capability. This capability may be granted by a system manager or account manager. (Optional parameter)

QUIET This parameter, if present, specifies that the message identifying the DS line number will be suppressed. The messages associated with subsequent REMOTE HELLO and REMOTE BYE commands will also be suppressed. The terminal operator is totally unaware that remote processing is taking place.

lusername is a user name on the local HP 3000, established by an account manager, that allows you to log-on under this account. This name is unique within the account. It contains from 1 to 8 alphanumeric characters, beginning with a letter. An at-sign (@) may be used to indicate the log-on user name. (Required parameter)

lacctname is the name of your account on the local HP 3000, as established by a system manager. It contains 1 to 8 alphanumeric characters, beginning with a letter. An at-sign (@) may be used to indicate the log-on account. (Required parameter)

NOTE: Must be preceded by period as delimiter.

lgroupname

is the name of a file group to be used for the local file domain and central-processor time charges, as established by an account manager. It contains from 1 to 8 alphanumeric characters, beginning with a letter. Default: Your home group if you are assigned one by an account manager. An at-sign (@) may be used to indicate the log-on group. (Optional if you have a home group; required if you do not.)

rusername

is a user name on the remote HP 3000 that allows you to access the account containing the remote data base. It follows the same rules as lusername. An at-sign (@) may be used to indicate rusername is same as lusername. (Required parameter)

racctname

is the name of the account on the remote HP 3000 that contains the data base. It follows the same rules as lacctname. An at-sign (@) may be used to indicate racctname is same as lacctname. (Required parameter)

rgroupname

is the name of the group on the remote HP 3000 to which the data base belongs. It follows the same rules as lgroupname. An at-sign (@) may be used to indicate rgroupname is same as lgroupname. (Optional parameter if home group for rusername is same as group containing data base; required if not.)

rupasw

is the password assigned to rusername. (Required if assigned)

rapasw

is the password assigned to racctname. (Required if assigned)

rgpasw

is the password assigned to rgroupname. (Not needed if rusername's home group contains data base. Otherwise, required if assigned.)

TIME=*cpusecs*

is the maximum central processor time that your remote session can use, entered in seconds. When this limit is reached, the remote session is aborted. It must be a value from 1 to 32767. To specify no limit, enter a question mark or omit this parameter. Default: No limit. (Optional parameter)

PRI=BS
CS
DS
ES

is the execution priority class that the Command Interpreter uses for your remote session, and also the default priority for all programs executed within the remote session. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest that the system permits for *racctname* or *rusername*, MPE assigns the highest priority possible below BS. Default: CS.
NOTE: DS and ES are intended primarily for batch jobs; their use for sessions is generally discouraged.

INPRI=*inputpriority*

is the relative input priority used in checking against access restrictions imposed by the job fence, if one exists. It takes effect at log-on time. It must be a value from 1 (lowest priority) to 13 (highest priority). If you supply a value less than or equal to the current job fence set by the console operator the session is denied access. Default: 8 if logging of session/job initiation is enabled, 13 otherwise. (Optional parameter)

HIPRI

is a request for maximum session-selection input priority, causing the remote session to be scheduled regardless of the current job fence or execution limit for sessions.
NOTE: You can specify this only if you have system manager or supervisor capability. (Optional parameter)

The following syntax should be noted:

- No spaces are allowed preceding the equal sign and between the equal sign and the HELLO command (=HELLO).
- Passwords are not allowed with the local user, account, and group names. They are not necessary since the local user passes the security password checks when logging on to the local session.
- The DSLINE parameter is not allowed in the HELLO command records since it is listed in Record 2.

FILENAME. After you have created the file with the Editor, you should KEEP it UNNumbered. The file name must follow the same rules as a data base name. It may be an alphanumeric string from 1 to 6 characters, the first character must be alphabetic.

USER IDENTIFICATION. Records 3 through n define a table that tells IMAGE which user, account, and group names on the local computer may access which user, account, and group names on the remote computer. You may specify remote user identification for more than one local user by creating a record for each local "user.account,group" in the format of Record 3 shown above. An at-sign symbol (@) may be substituted for any user, account, or group name in the record. If an at-sign is substituted for lusername, lacctname, or lgroupname, the name is replaced with the corresponding name specified at log-on time.

IMAGE searches for a match between the local user, account, and group names in the user table and the names used to log on to the local session. When a match has been found, and the local user, account, and group names are fully established, IMAGE looks for an at-sign to the right of =HELLO. If an at-sign is found, it is replaced with the corresponding name to the left of =HELLO. For example, if the record contains USERA.ACCTA,GROUPA=HELLO @.ACCTB,@, IMAGE replaces the first at-sign with USERA and the second with GROUPA. If an at-sign is not found, no substitutions are made. In either case, the information to the right of =HELLO is used as the remote log-on identification.

EXAMPLE. Suppose the following set of commands are stored in a data-base-access file:

Record 1	FILE STORE;DEV=DSL1#DISC
Record 2	DSLINE DSL1
Record 3	USERA.ACCTA,GROUPA=HELLO USERB.ACCTA,GROUPB
Record 4	@.ACCTA,GROUPA=HELLO USERA.ACCTA,GROUPA
Record 5	USERB.ACCTB,@=HELLO USERB.ACCTX,@

End of file

If a user logs on with the log-on identification indicated in the first column below, IMAGE will use the corresponding "user.account,group" identification in the second column to establish communication with the remote system.

	Log-on Identification		**Remote Identification Used**
User 1	USERA.ACCTA,GROUPA	⟶	USERB.ACCTA,GROUPB
User 2	USERB.ACCTA,GROUPA	⟶	USERA.ACCTA,GROUPA
User 3	USERB.ACCTB,GROUPB	⟶	USERB.ACCTX,GROUPB
User 4	USERA.ACCTB,GROUPB	⟶	None, no match found.

The first user's log-on identification matches the local user, account, and group names specified in Record 3, so the remote names specified in that record are used. The second user's account matches Record 3 but the user name does not, so IMAGE looks for another table entry with account ACCTA. Since the entry in Record 4 specifies any user (@) of ACCTA if their group is GROUPA, the second user's remote identification will be that specified in Record 4.

The third user logs on to ACCTB and a match is found in Record 5 since it specifies the same user name and accepts any group in the account.

The fourth user's account matches Record 5 but the user name does not match. Therefore, the fourth user cannot access the remote data base with this application program.

ACTIVATING A DATA-BASE-ACCESS FILE. After you have constructed a data-base-access file, you must use the DBUTIL utility program to activate the file. DBUTIL changes the file code to the IMAGE reserved code -402. Complete instructions for running the utility program are given in Section VI. Here is a summary of the operating instructions:

```
:RUN DBUTIL.PUB.SYS
        •
        •
>>ACTIVATE data-base-access file name
  Data-base-access file
    data-base-access file name is ACTIVATED

>>EXIT
```

The utility program checks that Record 1 contains a FILE command with the DEV=dsdevice parameter, Record 2 contains a DSLINE command with the same dsdevice value as the FILE command, and that all subsequent records contain =HELLO and no DSLINE parameter. If the file structure is inaccurate, the following message is displayed:

```
    Invalid contents of ascii access file
```

The utility program does not verify the accuracy of each command and its parameters. This is done at DBOPEN time and if, for example, the user.account identification is invalid, an error code for an unsuccessful DBOPEN is returned to the user's process.

DEACTIVATING A DATA-BASE-ACCESS FILE. In order to deactivate the data-base-access file, you use the DEACTIVATE command of the DBUTIL utility program. Complete instructions for this program are given in Section VI. Here is a summary of the operating instructions:

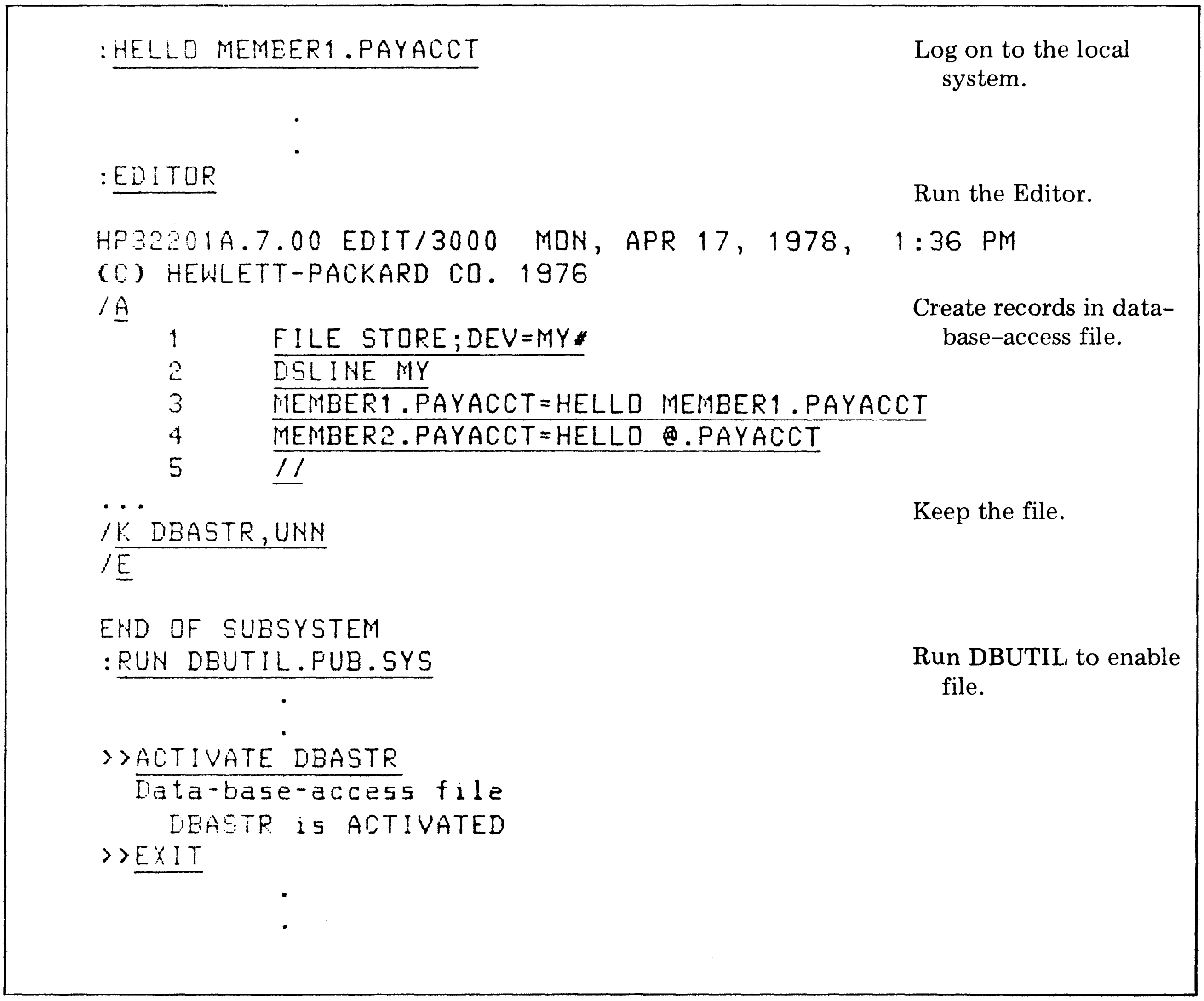
```
:RUN DBUTIL.PUB.SYS
        •
        •
>>DEACTIVATE data-base-access file name
  Data-base-access file
    data-base-access file name is DEACTIVATED

>>EXIT
```

You may want to do this in order to edit the content of the data-base-access file or prevent access through this file to the remote data base.

REFERENCING THE DATA BASE. To reference the data base from your local application program, use the data-base-access file name instead of the root file name when calling the IMAGE procedure. The word array specified as the *base* parameter must contain a pair of blanks followed by the left-justified data-base-access file name and terminated by a semicolon or blank (△). IMAGE recognizes the -402 file code and establishes a communications link to the remote HP 3000. If the data base is successfully opened, IMAGE replaces the pair of blanks with the extra data segment number of the assigned Remote Data Base Control Block. The *base* parameter must remain unchanged for the remainder of the process. When the application program calls the DBCLOSE procedure or terminates execution, an automatic REMOTE dsline#BYE and DSLINE;CLOSE is issued on the open DSLINE for that particular session.

EXAMPLE. The example in figure 8-5 illustrates how to create and activate a data-base-access file. In this case, the file named DBASTR is to be used to gain access to the STORE data base residing on a remote system in the PAYACCT account. The remote system is referenced by dsdevice name MY.

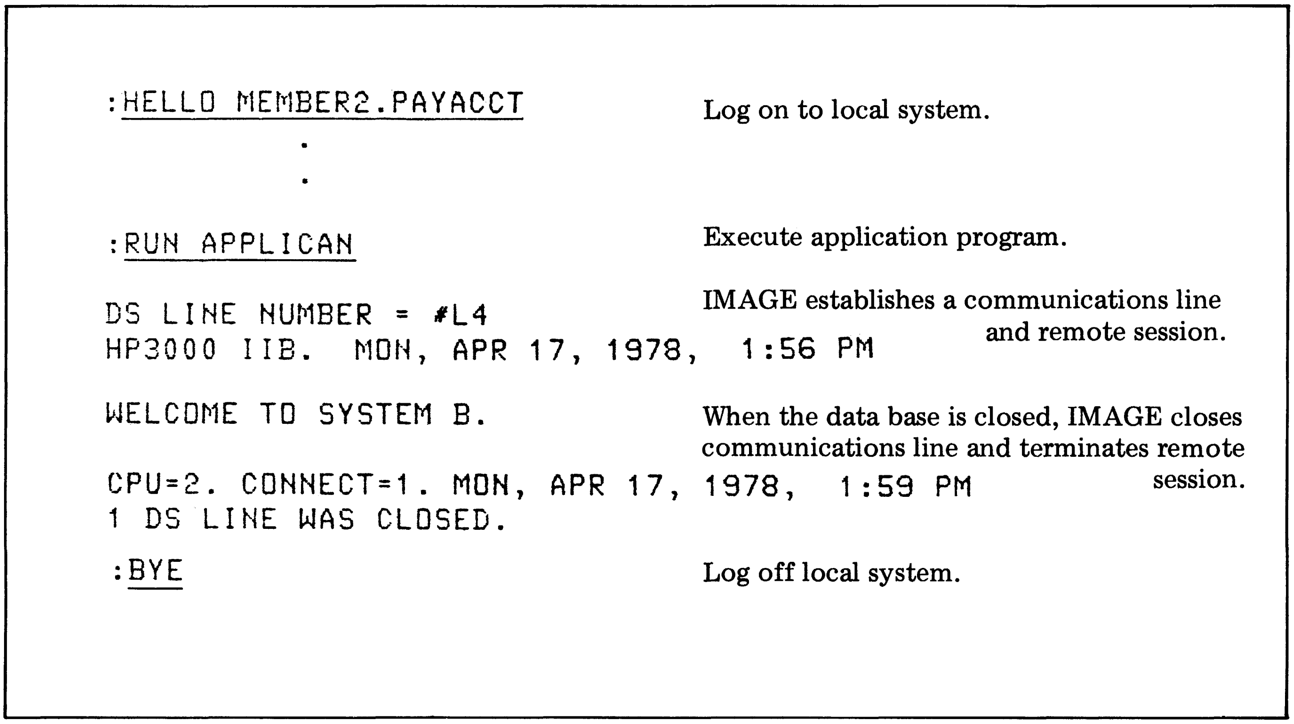
After the data-base-access file is created using the Editor, it is enabled by using the DBUTIL program.



```
:HELLO MEMEER1.PAYACCT                                 Log on to the local
                                                         system.
                .

                .
:EDITOR
                                                       Run the Editor.

HP32201A.7.00 EDIT/3000  MON, APR 17, 1978,  1:36 PM
(C) HEWLETT-PACKARD CO. 1976
/A                                                     Create records in data-
    1      FILE STORE;DEV=MY#                            base-access file.
    2      DSLINE MY
    3      MEMBER1.PAYACCT=HELLO MEMBER1.PAYACCT
    4      MEMBER2.PAYACCT=HELLO @.PAYACCT
    5      //
...                                                    Keep the file.
/K DBASTR,UNN
/E

END OF SUBSYSTEM
:RUN DBUTIL.PUB.SYS                                    Run DBUTIL to enable
                .                                        file.

                .
>>ACTIVATE DBASTR
  Data-base-access file
    DBASTR is ACTIVATED
>>EXIT
                .

                .
```

Figure 8-5. Preparing a Data-Base-Access File

Figure 8-6 illustrates use of the data-base-access file through a program named APPLICAN. After logging on to the local system, the user runs the program named APPLICAN from the local session. The *base* array in this program contains: △△ DBASTR. When a call to DBOPEN is executed, IMAGE establishes a communications line and remote session. When the program closes the data base, IMAGE closes the line and terminates the remote session.



```
:HELLO MEMBER2.PAYACCT
           .
           .
:RUN APPLICAN

DS LINE NUMBER = #L4
HP3000 IIB.  MON, APR 17, 1978,  1:56 PM

WELCOME TO SYSTEM B.

CPU=2. CONNECT=1. MON, APR 17, 1978,  1:59 PM
1 DS LINE WAS CLOSED.
:BYE
```

Log on to local system.

Execute application program.

IMAGE establishes a communications line and remote session.

When the data base is closed, IMAGE closes communications line and terminates remote session.

Log off local system.

Figure 8-6. Using a Data-base-access File

QUERY

When you use QUERY to retrieve information from a data base, you must specify a data base name, password and access mode before you can actually access the data base. The "DATA BASE=" prompt can be answered with a remote data base name or the data-base-access file name. A detailed description of QUERY/DS is provided in the *QUERY Reference Manual.*

ERROR MESSAGES

APPENDIX A

IMAGE issues three different types of error messages:

- Schema Processor Error Messages listed in tables A-1 through A-3
- Library Procedure Error Messages listed in tables A-4 through A-7
- Utility Error Messages listed in tables A-8 and A-9.

Schema Processor messages result from errors detected during processing of the data base schema. The library procedure messages consist of condition words returned to the calling program from the library procedures. The utility messages are caused by errors in execution of the data base utility programs.

SCHEMA PROCESSOR MESSAGES

The Schema Processor accesses three files:

- the textfile (DBSTEXT) containing the schema records and Schema Processor commands for processing.
- the listfile (DBSLIST) containing the schema listing, if requested, and error messages, if any.
- the root file, if requested, created as the result of an error-free schema.

Any file error which occurs while accessing any of these files causes the Schema Processor to terminate execution. A message indicating the nature of the error is sent to $STDLIST (and to the listfile, if listfile is different from $STDLIST).

Table A-1 lists the various file error messages. Each such message is preceded by the character string:

```
*****FILE ERROR*****
```

Additionally, the Schema Processor prints a standard MPE file information display on the $STDLIST file. Refer to the *MPE Intrinsics Reference Manual* or *Error Messages and Recovery Manual* for the meaning of MPE file information displays.

Schema Processor command errors may occur. They neither cause termination nor do they prohibit the creation of a root file. In some cases, however, the resultant root file will differ from what might have occurred had the commands been error free. Command errors are added to an error count which, if it exceeds a limit (see Section III), will cause the Schema Processor to terminate execution.

Table A-2 lists the various Schema Processor command error messages. Each such message is preceded by the character string:

```
*****ERROR*****
```

Data base definition syntax errors may be detected by the Schema Processor. Their existence does not cause termination but does prohibit root file creation. Discovery of one may trigger others which disappear after the first is corrected. Also, detection of one may preclude detection of others which appear after the first is corrected. Syntax errors are also added to an error count which, if excessive, will cause Schema Processor termination.

Table A-3 lists the various syntax error messages. As with command errors, each syntax error is preceded by the character string:

    *****ERROR*****

If the LIST option is active (see Section III), error messages for command errors and syntax errors appear in the listfile following the offending statement. If the NOLIST option is active, only the offending statement, followed by the error message, is listed.

Table A-1. IMAGE Schema Processor File Errors

MESSAGE	MEANING	ACTION
READ ERROR ON file name	FREAD error occurred on the specified file.	Check *textfile* or :FILE command.
UNABLE TO CLOSE file name	FCLOSE error occurred on specified file. May be caused by duplicate file in group with same name as root file.	Change data base name or purge file of same name. or
UNABLE TO USE file name	Specified file cannot be FOPENed or its characteristics make it unsuitable for its intended use.	Be sure correct file and file name used. Check :FILE commands used. If other cause, consult MPE Intrinsics Reference Manual for similar message.
UNABLE TO WRITE LABEL OF file name	FWRITELABEL error occurred on specified file.	
UNEXPECTED END-OF-FILE ON file name	Call to FREAD or FWRITE on specified file has yielded unexpected end of file condition	
WRITE ERROR ON file name	FWRITE error occurred on the specified file.	

Table A-2. IMAGE Schema Processor Command Errors

MESSAGE	MEANING	ACTION
COMMAND CONTINUATION NOT FOUND	If schema processor command is continued to next record, the last non-blank character of preceding line must be an ampersand (&), and continuation record must start with dollar sign ($).	Examine schema textfile to find incorrect command, edit, run Schema Processor again.
COUNT HAS BAD FORMAT	Numbers appearing in ERRORS, LINES, BLOCKMAX parameters of the $CONTROL command are not properly formatted integer values.	
ILLEGAL COMMAND	Schema processor does not recognize the command. Valid commands are $PAGE, $TITLE, and $CONTROL.	
IMPROPER COMMAND PARAMETER	One of the parameters in a command is not valid.	
MISSING QUOTATION MARK	Character string specified in $PAGE or $TITLE command must be bracketed by quotation marks (").	
SPECIFIED TITLE TOO LONG	Character string appearing in $TITLE or $PAGE command exceeds 104 characters.	

Table A-3. IMAGE Schema Syntax Errors

MESSAGE	MEANING	ACTION
AUTOMATIC MASTER MUST HAVE SEARCH ITEM ONLY	AUTOMATIC master data sets must contain entries with only one data item which must be a search item.	Examine schema to find incorrect statement, edit, and run Schema Processor again.
BAD CAPACITY OR TERMINATOR	Either the number in CAPACITY: statement is not an integer between 1 and 2^{23} –1 or a semicolon is missing.	
BAD PATH COUNT OR TERMINATOR	Path count in master data set definition is not an integer 1 to 16 (for an automatic) or 0 to 16 (for a manual). This message may also indicate path count is not followed by ")".	
BAD CHARACTER IN USER CLASS NUMBER	User class number in password part is not integer from 1 to 63.	
BAD DATA BASE NAME OR TERMINATOR	Data base name in BEGIN DATA BASE statement is not valid data base name of from 1 to 6 alphanumeric characters beginning with an alphabetic, or is not followed by semicolon.	
BAD DATA SET TYPE	Data set type designator is not AUTOMATIC (or A), MANUAL (or M), or DETAIL (or D).	
BAD PATH CONTROL PART DELIMITER	Data item defined as sort item in detail data set is not properly delimited with parentheses.	
BAD PATH SPECIFICATION DELIMITER	Name of master data set following search item name in a detail data set definition is not followed by a ")" or by a sort item name enclosed in parentheses.	
BAD READ CLASS OR TERMINATOR	Read user class number defined for either a data set or data item is not an integer from 0 to 63 or is not terminated by a comma or slash.	
BAD SET NAME OR TERMINATOR	Data set name does not conform to rules for data set names (1 to 16 alphanumeric characters beginning with alphabetic, chosen from the set: A-Z, 0-9, or + – * / ? ' # % & @), or is not terminated by correct character for context in which it appears.	
BAD SUBITEM COUNT OR TERMINATOR	Subitem count for a data item defined in schema item part is not an integer from 1 to 255.	

Table A-3. IMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
BAD SUBITEM LENGTH OR TERMINATOR	Subitem length for data item defined in schema item part is not an integer from 1 to 255.	Examine schema to find incorrect statement, edit, and run Schema Processor again.
BAD TERMINATOR – ';' OR ',' EXPECTED	Items within an entry definition must be separated from each other by commas and terminated by a semicolon.	
BAD TERMINATOR – ';' EXPECTED	Password or capacity was not followed by a semicolon.	
BAD TYPE DESIGNATOR	Data item defined in schema item part is not defined as type I, J, K, R, U, X, Z, or P.	
BAD WRITE CLASS OR TERMINATOR	Write user class number indicated for either data set or data item is not an integer from 0 to 63 or is not terminated by right parentheses or a comma.	
'CAPACITY:' EXPECTED	CAPACITY statement must follow entry definition in definition of data set in set part of schema.	
DATA BASE HAS NO DATA SETS	No data sets were defined in set part of schema. Data base must contain at least one data set.	
DUPLICATE ITEM SPECIFIED	Same data item name used more than once in entry definition of data set.	
DUPLICATE ITEM NAME	Data item name appears more than once in item part of schema.	
DUPLICATE SET NAME	Same name has been used to define more than one data set in schema set part.	
'ENTRY:' EXPECTED	Each data set defined in schema set part must contain ENTRY: statement followed by data item names of data items in entry.	
ENTRY TOO BIG	Number and size of data items defined for an entry of a data set yields entry which is too large for maximum block size (as either specified by $CONTROL BLOCKMAX= command or by default).	
ENTRY TOO SMALL	Detail data set that is not linked to any master data set must have data entry length equal to or greater than two words. This length is determined by adding size in words of each data item defined in data entry.	

Table A-3. IMAGE Schema Syntax Errors (Continued)

MESSAGE	MEANING	ACTION
ILLEGAL USER CLASS NUMBER	User class number defined in schema password part is not an integer between 1 and 63 inclusive.	Examine schema to find incorrect statement, edit, and run Schema Processor again.
ILLEGAL ITEM NAME OR TERMINATOR	Data item name does not conform to naming rules. (1 to 16 alphanumeric characters beginning with alphabetic, chosen from the set: A-Z, 0-9, or + – * / ? ' # % & @), or if in the item part, is not followed by a comma.	
ITEM LENGTH NOT INTEGRAL WORDS	Data item, simple or compound, must occupy an integral number of words.	
ITEM TOO LONG	Single data item cannot exceed 2047 words in length.	
MASTER DATA SET LACKS EXPECTED DETAILS	Master data set was defined with a non-zero path count, but the number of detail search items which back-referenced master is less than value of path count.	
MASTER DATA SET LACKS SEARCH ITEM	Master data set was defined without defining one of the data items in the set as a search item.	
MORE THAN ONE KEY ITEM	Master data set cannot be defined with more than one search item.	
MORE THAN ONE PRIMARY MASTER	User has defined more than one primary path for a detail data set.	
'NAME:' or 'END.' EXPECTED	Schema processor expected at this point to encounter the beginning of another data set definition or end of schema.	
'PASSWORDS:' NOT FOUND	'PASSWORDS:' statement must immediately follow the BEGIN DATA BASE statement in schema. If it does not, DBSCHEMA terminates execution.	
PASSWORD TOO LONG	Any password defined in data schema cannot exceed eight characters	
REFERENCED SET NOT A MASTER	Data set referenced by detail data set search item is another detail data set rather than a master.	

Table A-3. IMAGE Schema Syntax Error (Continued)

MESSAGE	MEANING	ACTION
SCHEMA PROCESSOR LACKS NEEDED TABLE SPACE	Schema processor is unable to expand its data stack to accommodate all of the translated information which will make up the root file. It continues to scan the schema for proper form, but will not perform all of the checks for correctness nor create a root file. To process schema correctly, operating system must be configured with a larger maximum stack size.	Ask system manager to increase maximum stack size.
SEARCH ITEM NOT SIMPLE	All data items defined in data schema as search items must be simple items	Examine schema to find incorrect statement, edit, and run Schema Processor again.
SEARCH ITEMS NOT OF SAME LENGTH	Master search item must be same type as any related detail data set search item.	
SEARCH ITEMS NOT OF SAME TYPE	Master search item must be same type as any related detail data set search items.	
SET HAS NO PATHS AVAILABLE	More detail data set search items have specified a relationship with a master data set than the number specified in master data set's path count.	
SORT ITEM NOT IN DATA SET	Detail data set's entry definition does not include an item which is specified as sort item for another item in entry.	
SORT ITEM OF BAD TYPE	Data item defined as sort item must be of type U, K, or X.	
SORT ITEM SAME AS SEARCH ITEM	Same item cannot be both search and sort item for same path.	
TOO MANY DATA ITEMS	Item part of schema may contain no more than 255 data item names.	
TOO MANY DATA SETS	Data base can contain no more than 99 data sets.	Check schema, edit, run again.
TOO MANY ERRORS	Specified or default maximum number of errors has been exceeded. Processing terminates.	Correct errors and/or increase ERROR parameter value.
TOO MANY ITEMS SPECIFIED	Data set entry can contain no more than 127 data items	Examine schema to find incorrect statement, edit, and run Schema Processor again.
TOO MANY PATHS IN DATA SET	Detail data set entry can contain no more than 16 search items.	
UNDEFINED ITEM REFERENCED	Data item appearing in data set definition was not previously defined in item part of data schema.	
UNDEFINED SET REFERENCED	Master data set referenced by detail search item was not previously defined in set part of data schema.	

LIBRARY PROCEDURE ERROR MESSAGES

The success of each call to an IMAGE library procedure is reflected upon return to the user by the hardware condition code and the value of a condition word returned in the first word of the status area.

If the procedure fails to execute properly, the hardware condition code is set to CCL (Condition Code Less) and the procedure returns a negative integer in the condition word. Table A-4 describes the negative condition words resulting from file system and memory management failures, while Tables A-5 and A-5a describe the negative condition words resulting from calling errors and communications errors respectively.

If the procedure operates properly but encounters an exceptional condition, such as end-of-file, the hardware condition code is set to CCG (Condition Code Greater) and the procedure returns a positive integer in the condition word. Table A-6 describes the positive condition words resulting from exceptional conditions.

If the procedure operates properly and normally, the hardware condition code is set to CCE (Condition Code Equal) and the procedure returns zero in the condition word.

In addition to returning a condition word, all IMAGE library procedures put information about the procedure call into the fifth through tenth words of the status area. This information may be useful in debugging your programs, because it describes the conditions in which the particular results are obtained. This information is used by DBEXPLAIN and DBERROR when they are interpreting the results of IMAGE calls.

In a few cases this information is not returned by the IMAGE procedure because it uses the same words in the status area for returning other data. Specifically, successful execution of DBFIND, DBGET, DBUPDATE, DBPUT, or DBDELETE puts the other information here as described in Section IV of this manual.

For any other return, error or non-error, from a library procedure, the specified words of the status area have the following contents:

Word	Contents
5	The PB-relative offset within the calling program's code segment of the current procedure call to the IMAGE library procedure (the location of the PCAL instruction in the SPL code.)
6	Bits 7-15: The intrinsic number of the called IMAGE library procedure. Bits 0-3: Zero or the access mode in which the data base is opened.
7	The DB-relative word address of the *base* parameter.
8	The DB-relative word address of the *password*, *qualifier*, or *dset* parameter.
9	The value of the *mode* parameter.
10	The PB-relative offset within the library procedure code segment at which return to the calling program was initiated (for HP use only).

Consult the *MPE Commands Reference Manual* for a discussion of PB and DB registers, and the *Systems Programming Language Reference Manual* for more information about the PCAL instruction.

ABORT CONDITIONS

In general, four types of error conditions can cause IMAGE to abort the calling process:

1. A call from a user process with the hardware DB register not pointing to the process stack.

2. A faulty calling sequence.

3. An internal error in an MPE file intrinsic which the calling procedure cannot correct.

4. An internal inconsistency in the data base or Data Base Control Block discovered by a library procedure.

In case 1, the procedure prints the standard MPE run-time abort message described in Section II, of the *Error Messages and Recovery Manual.* In cases 2, 3, and 4, IMAGE prints additional information on the standard list device about the error prior to printing the standard MPE abort message. The first line of this information is:

ABORT: *procedure name* ON DATA BASE *name;*

where *procedure name* is the name of the library procedure which caused the abort and *name* is the name of the data base being accessed at the time of the abort. Table A-7 describes additional lines of information which may appear prior to the standard MPE abort message.

Some of the abort conditions are due to an error in one of the MPE file intrinsics FOPEN, FREADLABEL, FREADDIR, FWRITELABEL, FWRITEDIR, or FCLOSE. Aborts of this type generally occur after the procedure has possibly altered the data base so that the data base structure has been damaged in some way. Each of the messages in table A–7 which refer to an IMAGE data file is followed by an MPE file information display which lists all of the characteristics of the MPE data set or root file where the error occurred, along with an MPE error number. For more information about file error codes consult Section II of the *Error Messages and Recovery Manual* and for the file information display, consult appendix A of that manual.

Table A-4. IMAGE Library Procedure File System and Memory Management Errors

CCL	PROCEDURE	MEANING	ACTION
-1	MPE intrinsic FOPEN failure	For DBOPEN, error may indicate that data base could not be opened. Possible reasons: ● Data base name string not terminated with semicolon or blank ● Data base does not exist or is secured against access by its group or account security ● Data base is already opened exclusive or in mode incompatible with requested mode ● MPE file system error occurred. For DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE, error may occur if: ● The process has too many files open external to the data base ● Data set does not exist or is secured against access ● Some other MPE file system error has occurred.	Determine which of probable causes applies and either modify application program or see system manager about file system error.
-2	MPE intrinsic FCLOSE FAILURE	This is an exceptional error (should never happen) and is returned only by DBOPEN or DBCLOSE. Indicates a hardware or system software failure.	Notify system manager of error.
-3	MPE intrinsic FREADDIR failure	This is an exceptional error (as -2 above) and is returned by DBOPEN, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE	
-4	MPE intrinsic FREADLABEL failure	This is an exceptional error (as -2 above) and is returned by DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE.	

NOTE: For condition words -1 through -4, second word of calling program's status area is the data set number for which file error occurred (zero indicates root file). Third word is MPE failure code returned by FCHECK intrinsic. Refer to *Error Messages and Recovery Manual*, Section II, for meaning of this code.

Table A-4. IMAGE Library Procedure File System and Memory Management Errors (Continued)

CCL	PROCEDURE	MEANING	ACTION
-9	MPE intrinsic GETDSEG failure.	This is an exceptional error and is returned by DBOPEN when it cannot obtain extra data segment for use as Data Base Control Block. This occurs if required virtual memory space is unavailable or if required DST entry is unavailable. Second word of user's status area is size (in words) of data segment which memory management was unable to supply. Third word is MPE failure code returned by GETDSEG intrinsic.	Notify system manager of problem or wait until system is less busy.

Table A-5. IMAGE Library Procedure Calling Errors

CCL	CONDITION	MEANING	ACTION
-11	Bad *base* parameter	For DBOPEN, the first two characters in *base* are not blank, or data base name contains special characters other than period. For all other procedures, either first two characters in *base* do not contain the value assigned by DBOPEN, or exceptionally, the parameters passed to procedure are incorrect in type, sequence or quantity.	Check application program's procedure call. Correct error in call.
-12	No covering lock	For DBUPDATE, DBPUT, and DBDELETE, data base has been opened in DBOPEN Mode 1 but there is no lock to cover entry in question. DBPUT or DBDELETE to master requires data set or data base be locked. In all other cases, entry, set, or data base can be locked.	Modify program to apply proper lock or change mode.
-14	Illegal intrinsic in current access mode	For DBPUT and DBDELETE data base has been opened in DBOPEN Mode 2, 5, 6, 7, or 8. These procedures may not be used with these access modes. For DBUPDATE, data base has been opened in DBOPEN Mode 5, 6, 7, or 8. DBUPDATE may not be used with these modes.	Modify program. Alter either mode or procedure call or notify current user that operation cannot be performed.
-21	Bad password	For DBOPEN, user class granted does not permit access to any data in data base. This is usually due to incorrect or null password.	Supply correct password.

Table A-5. IMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-21	Bad data set reference	For DBINFO (modes 104, 201, 202, 301, and 302), DBCLOSE, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE, when data set reference is: ● Numeric but out of range of the number of data sets in data base ● An erroneous data set name ● A reference to data set which is inaccessible to user class established when data base opened. For DBFIND, this error is also returned if referenced data set is a master. Erroneous data set name may arise when a terminating semicolon or blank is omitted.	Check application program's procedure call. Correct error in call.
-21	Bad data item reference	For DBINFO (modes 101, 102, and 204), data item reference is: ● Numeric but out of range of the number of data items in data base ● An erroneous data item name ● A reference to data item which is inaccessible to user class established when data base opened. An erroneous data item name may arise when a terminating semicolon or blank is omitted.	Check application program's procedure call. Correct error in call.
-23	Data set not writable	For DBPUT and DBDELETE, data base has been opened in DBOPEN Mode 1, 3, or 4 and user has read but not write access to the referenced data set.	Modify access mode set in procedure call or notify current user operation cannot be performed.
-24	Data set is an automatic master	For DBPUT, the referenced data set is an automatic master.	Modify data set name in call or in data set type in schema.
-31	Bad *mode*	This error occurs in all procedures when the *mode* parameter is invalid. For DBGET, *mode* is 7 or 8 and referenced data set is a detail, or *mode* is 5 or 6 and referenced data set is a detail without search items.	Correct mode in procedure call.
-32	Unobtainable *mode*	For DBOPEN, root file cannot be FOPENed with access options (AOPTIONS) required for the specified *mode*. Second word of calling program's status area is required	See the *MPE Intrinsics Reference Manual* for meaning of AOPTIONS words.

Table A-5. IMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-32	(continued)	AOPTIONS, and third word is the AOPTIONS granted to DBOPEN by MPE file system. This error usually occurs either due to concurrent data base access by other users or due to MPE account or group security provisions.	Action depends on program's design. Normally notify user that requested access mode is not available.
-51	Bad *list* length	For DBGET, DBUPDATE, and DBPUT, the list is too long. This may occur if list is not terminated with a semicolon or blank. It may also occur for otherwise legitimate lists which are too long for IMAGE's work area. It will never occur for numeric lists.	Shorten *list* array contents. If necessary, change to numeric list.
-52	Bad *list* or bad *item*	For DBGET, DBUPDATE, or DBPUT, the *list* parameter is invalid. *list* either has a bad format or contains a data item reference which: ● Is out of range of the number of data items in the data base. ● Reference an inaccessible data item. ● Duplicates another reference in the list. For DBFIND, the *item* parameter contains data item reference which either: ● Is out of range of the number of data items in the data base. ● Is not a search item for referenced data set.	Check procedure call. Correct error in call or parameter.
-53	Missing search or sort item	For DBPUT, a search or sort item of referenced data set is not included in *list* parameter.	Check procedure call. Correct error in call or parameter.
-91	Bad root modification level	For DBOPEN, the software version of the DBOPEN procedure is incompatible with version of schema processor which created root file.	Check with system manager that you have correct IMAGE software. If necessary ask HP support personnel about conversion.

Table A-5. IMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-92	Data base not created	For DBOPEN, the referenced data base has not yet been created and initialized by the DBUTIL program (in CREATE mode).	Run DBUTIL to create data base. Try application program again.
-94	Data base bad	For DBOPEN, the referenced data base is flagged "bad", and cannot be opened by programmatic calls to DBOPEN. A "bad" data base is structurally damaged, probably as a result of a system crash during DBLOAD or some other "output deferred" operation.	Purge the data base and restore a backup copy, or erase the data base and reload it. You may be able to salvage the current contents by first performing a serial DBUNLOAD.
-120	Not enough stack to perform DBLOCK	DBLOCK cannot obtain enough stack space.	:RUN or :PREP with STACK= or MAXDATA= to get more stack space.
-121	Descriptor count error	DBLOCK detected an error in the descriptor count (first word of qualifier array) in locking mode 5 or 6.	Count must be a positive integer.
-122	Descriptor list bad. Is not entirely within stack.	DBLOCK checked the list and found that it did not lie between DL and the top of stack. May be caused by a bad *length* field.	Check length of each descriptor, and descriptor count.
-123	Illegal *relop* in a descriptor.	DBLOCK encountered a *relop* field containing characters other than > =, < =, = Δ or Δ=.	Check contents of *qualifier* array.
-124	Descriptor too short. Must be greater than or equal to 9.	DBLOCK encountered a lock descriptor less than 9 words long.	
-125	Bad set name/number.	DBLOCK *qualifier* array contains an invalid data set name or number. (Refer to error -21 for rules.)	Check contents of qualifier array. Be sure names delimited by semicolon or space if less than 16 bytes long.
-126	Bad item name/number.	DBLOCK *qualifier* array contains an invalid data item name or number. (Refer to error -21 for rules.)	
-127	Attempt to lock using a compound item.	DBLOCK does not allow compound items in lock descriptors.	Modify locking strategy to lock on a non-compound item.
-128	Value field too short in a descriptor.	A *value* field in a DBLOCK lock descriptor must be at least as long as the data item for which it is specified.	Perhaps the length word is too small in the descriptor.

Table A-5. IMAGE Library Procedure Calling Errors (Continued)

CCL	CONDITION	MEANING	ACTION
-129	P-type item longer than P28 specified.	DBLOCK does not allow P-type data items longer than 28 in lock descriptors. (27 digits plus sign.)	Modify locking strategy to lock on a different item.
-130	Illegal digit in a P-type value.	DBLOCK has encountered a P-type value in a lock descriptor with an invalid packed decimal digit.	Check *qualifier* array contents to determine why data is invalid. Correct data representations are described in the *Machine Instruction Set Manual* Section III.
-131	Lowercase character in type-U value.	DBLOCK has encountered a lowercase character in a type-U value specified in a lock descriptor.	
-132	Illegal digit in type Z value.	Lock descriptor value specified to DBLOCK contains an invalid zoned decimal digit.	
-133	Illegal sign in type Z value.	Lock descriptor value specified to DBLOCK contains an invalid zoned decimal sign.	
-134	Two descriptors conflict.	DBLOCK has detected two lock descriptors in the same call that lock the same or part of the same data base entity. (For example, lock on set and data base in same request.	Check *qualifier* array contents for conflicting lock descriptors.
-135	Second lock without CAP=MR	A second call to DBLOCK has been made without an intervening DBUNLOCK call and program does not have MR capability.	Read discussion of multiple calls to DBLOCK in Section IV of this manual if you plan to use CAP=MR.
-136	Descriptor list exceeds 2047 words.	DBLOCK allows at most 2047 word long lock descriptor lists (*qualifier* array).	Change *qualifier* array contents so lock descriptor list is shorter.

Table A-5a. IMAGE Library Procedure Communications Errors

CCL	CONDITION	MEANING	ACTION
-60	Illegal file equation on root file	When using a :FILE command with the data base name or a data-base-access file name, only the file designators and DEV= parameters are allowed.	Re-enter :FILE command without illegal parameters.
-100	DSOPEN failure	While executing a DBOPEN, IMAGE has encountered a hardware failure trying to obtain a communications line.	Try opening the data base again. If error persists contact your HP Customer Engineer.
-101	DSCLOSE failure	This is an exceptional error returned by DBOPEN or DBCLOSE. It indicates a hardware or system software failure.	Notify system manager of problem.
-102	DSWRITE failure	A line failure has occurred while attempting an operation on a remote data base. May be returned by DBOPEN, DBFIND, DBGET, DBPUT, DBUPDATE, DBDELETE, DBLOCK, DBUNLOCK, DBINFO, or DBCLOSE.	Try calling the procedure again. If error persists notify system manager.
-103	Remote stack too small	Command Interpreter on remote HP 3000 cannot obtain stack space necessary to execute a DBOPEN or DBLOCK.	Ask system manager of remote system to increase available stack size.
-104	Remote system does not support IMAGE	Remote HP 3000 does not contain IMAGE software. May be returned by DBOPEN.	Ask system manager of remote system to obtain and load IMAGE software.
-105	MPE intrinsic GETDSEG failure on remote HP 3000	This is an exceptional error and is returned by DBOPEN on the remote system when it cannot obtain an extra data segment for the DBCB. Second word of status array is size of data segment.	Notify system manager of problem.
-106	Remote data inconsistent	This is an exceptional error returned by same intrinsics as -102 (see above). It indicates a hardware or system software failure.	Notify system manager of problem.
-107	DS procedure call error	This is an exceptional error returned by same intrinsics as -102 (see above). It indicates a hardware or system software failure.	Notify system manager of problem.

NOTE: For condition codes -100 through -102, third word of calling program's status area is MPE failure code returned by DSCHECK intrinsic.

Table A-6. IMAGE Library Procedure Exceptional Conditions

CCG	CONDITION	MEANING	ACTION
10	Beginning of file	DBGET has encountered beginning of file during a backward serial read. (There are no entries before the one previously accessed.)	Appropriate action depends on program design.
11	End of file	DBGET has encountered the end of file during a forward serial read. (There are no entries beyond the most recently accessed one.)	
12	Directed beginning of file	DBGET has been called for a directed read with a record number less than 1.	
13	Directed end of file	DBGET has been called for a directed read with a record number greater than the capacity of data set.	
14	Beginning of chain	DBGET has encountered beginning of chain during a backward chained read.	
15	End of chain	DBGET has encountered end of chain during a forward chained read.	
16	Data set full	DBPUT has discovered that data set is full.	Restructure data base with larger capacity for this data set. See Section VI.
17	No master entry	DBFIND is unable to locate master data set entry (chain head) for specified detail data set's search item value.	Appropriate action depends on program design.
17	No entry	DBGET has been called to reread an entry, but no "current record" has been established or a call to DBFIND has set the current record to 0. DBGET is unable to locate master data set entry with specified search item value. DBGET has discovered that selected record is empty (does not contain an entry). DBUPDATE or DBDELETE was called when the "current record" was not established or was empty.	
18	Broken chain	For DBGET with *mode* parameter equal to 5 (forward chained read), the "next entry" on current chain (as designated by internally maintained forward pointer for data set) contains backward pointer which does not point to most recently accessed entry (or zero for first member of a chain).	Begin reading chain again from first or last entry.

Table A-6. IMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
18	(continued)	For DBGET with *mode* parameter equal to 6 (backward chained read), the "next entry" on current chain in a backward direction (as designated by internally maintained backward pointer for data set) contains a forward pointer which does not point to most recently accessed entry (or zero for last entry in a chain). This error can arise in DBOPEN access modes 1, 5, and 6 because another user can make data base modifications concurrent with this user's accesses. When this error occurs, no data is moved to user's stack, although internal pointers maintained by IMAGE in the DSCB are changed to new "offending" entry. (It becomes the current entry.) Note that this error check does not detect all structural changes. DBGET makes check only when preceding call on data set was successful DBFIND or DBGET.	
20	Data base locked or contains locks.	DBLOCK (in conditional mode) has discovered that whole data base cannot be locked. *Status* word 3 explains why. It equals 0 if data base locked, equals 1 if data base contains locked data sets or entries. Returned in DBLOCK mode 2 only.	Appropriate action depends on program design.
22	Data set locked by another process.	DBLOCK has detected that the data set is locked by another process or this process through a different access path. Returned in DBLOCK modes 4 and 6 only.	
23	Entries locked within set.	DBLOCK has detected that data entries within requested data set are locked by another process or this process through a different access path. Returned in DBLOCK mode 4 only.	
24	Item conflicts with current locks.	Lock descriptors passed to DBLOCK specify a data item that is different than one used to set existing locks. IMAGE allows no more than one data item per data set to be used at one time for locking purposes. Returned in DBLOCK mode 6 only.	
25	Entries already locked.	DBLOCK has detected that data entries requested to be locked are already locked by another process or this process through a different access path. Returned in DBLOCK mode 6 only.	

Table A-6. IMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
41	Critical item	DBUPDATE has been asked to change value of search or sort item.	Correct call or notify user cannot update item.
42	Read only item	DBUPDATE has been asked to change value of a data item for which the user does not have write access.	Notify user, cannot update item. Or change password in program.
43	Duplicate search item value.	DBPUT has been asked to insert data entry into a master data set with a search item value which already exists in data set.	Appropriate action depends on program design.
44	Chain head	DBDELETE has been asked to delete master data set entry which still has one or more non-empty chains.	
50	Buffer too small	Calling program's buffer (identified by *buffer* parameter) is too small for amount of information that DBGET or DBINFO wishes to return.	Correct procedure call, or change buffer name or size.
51	Insufficient stack for BIMAGE temporary buffer	The stack size is not large enough for the temporary buffer used by the BASIC IMAGE interface routines: XDBGET, SDBPUT, XDBUPDATE, and XDBINFO.	Ask system manager to increase maximum stack size.
52	Invalid number of parameters	Call to BIMAGE interface procedure has either too many or too few parameters.	Correct procedure call.
53	Invalid parameter	Call to BIMAGE interface procedure has an invalid parameter, for example, a parameter of wrong type.	Correct parameter name in call or parameter itself.
54	Status array too small	The status array specified in call to BIMAGE interface procedure has less than 10 elements	Dimension status array with 10 elements.
61	This data base opened more than 63 times by same process	DBOPEN has been called when the specified data base has already been opened 63 times by the same process.	
62	DBCB full	IMAGE is unable to expand the trailer area in the DBCB by enough to process a DBGET, DBPUT, or DBUPDATE *list*. Or, if DBLOCK returned condition: Lock area within DBCB is full or system would not allow IMAGE to expand DBCB, or trailer could not be expanded to hold descriptor list.	

Table A-6. IMAGE Library Procedure Exceptional Conditions (Continued)

CCG	CONDITION	MEANING	ACTION
63	Bad DBCB	Another process sharing data base has aborted because of logical inconsistency or internal error in iMAGE, leaving DBCB in potentially inconsistent state. All user accesses through existing DBCB are disabled (except for DBCLOSE, mode 1). Returned by all intrinsics.	
64	PCBX data segment area full	DBOPEN is unable to open data base because there is no room for DBCB entry in PCBX area (MPE portion of data stack).	
66	The current DBCB for the data base does not appear correct (IMAGE internal error)		
1xx	Missing chain head	User has attempted to add detail data entry with a search item value that does not match any existing search item value in corresponding manual master data set. The digits xx identify the offending path number established by order in which their search items occur in set part of schema.	Notify user cannot add entry or add manual master entry and try again.
2xx	Full chain	User has attempted to add detail data entry to a chain which already contains the maximum allowable (65535) entries. The digits xx identify the offending path number (as described in 1xx).	Consult with data base manager. May need to delete some entries from chain or restructure data base.
3xx	Full master	User has attempted to add detail data entry with a search item value in corresponding automatic master data set and new master entry cannot be created because automatic master data set is full. xx is offending path number, (as described in 1xx).	Restructure data base, increasing capacity of automatic master. (See Section VI).

Table A-7. IMAGE Library Procedure Abort Condition Messages

MESSAGE	MEANING	ACTION
BUFFER SUPPLY CRISIS	Internal software inconsistency has caused IMAGE to improperly manage its buffer space.	Notify system manager and possibly HP support personnel of error. Save FID information if printed. It may be necessary to perform data base recovery procedures. (See Section VI).
CRITICAL LABEL READ ERROR ON data set	Procedure was unable to read label of data base file.	
CRITICAL READ ERROR ON data set	Procedure encountered MPE file read error while reading data base file.	
LABEL WRITE ERROR ON data set	Procedure was unable to complete the writing of user label of data base file.	
LOST FREE SPACE IN data set	Internal software inconsistency has caused unused record locations in data set to become lost or unavailable. IMAGE prints out a file information display for the data set file.	
NEGATIVE MOVE ATTEMPT: n	Internal software inconsistency has been detected by IMAGE while attempting to move data to or from user's stack.	
UNABLE TO CLOSE data set	Procedure was unable to close a data base file.	
UNABLE TO OPEN data set	Procedure was unable to open a data base file.	
WRITE ERROR ON data set	Procedure encountered an MPE file write error while writing into data base file.	
WRONG NUMBER OF PARAMETERS OR BAD ADDRESS FOR PARAM *#n*	Address referenced by one of the parameters is not within user's stack area in memory (roughly between the DL and Q registers). n is positional number of the parameter in procedure's calling sequence. First parameter is number 1, second is number 2, etc.	

UTILITY ERROR MESSAGES

Two types of error messages are generated by the Utility programs. The first type consists of conditional errors associated with accessing the desired data base. For all utility programs except DBUTIL, errors generating these messages can be corrected without terminating the run if you are in session mode. DBUTIL errors of this type may terminate the program. After printing the error message the DBUTIL program reprompts with two greater than symbols (>>). Other utilities reprompt with the message: "WHICH DATA BASE?", allowing you to re-enter the data base reference. If you wish to terminate the utility program at this point you may type a carriage return with or without leading blanks. If you are in job mode, conditional errors always cause program termination. Conditional error messages and their meanings are described in table A-8.

Unconditional errors occur in utility programs after successful execution has already begun. These errors usually cause program termination. The accompanying messages and their meanings are described in table A-9.

Certain errors, external to the utilities, can result in utility program termination. Those caused by the operating system or initiated by the console operator are explained in Section II or Section VII of the *Error Messages and Recovery Manual*. Errors initiated by the library procedures called by the utilities are described in tables A-4 through A-7 of this manual.

Table A-8. IMAGE Utility Program Conditional Messages

MESSAGE	MEANING	ACTION
BAD DATA BASE REFERENCE	Data base reference following the utiiity program :RUN command contains syntax error.	In session mode, correct error or press *return* to terminate program.
BAD MAINTENANCE WORD	User invoking utility is not the creator of the referenced data base and has supplied incorrect maintenance word.	
Cannot open data base	Data base cannot be opened at this time. It may already be open in a mode that does not allow concurrent access.	Try DBUTIL command again later.
Data-base-access file does not exist	No such file exists in log-on group or account.	Check files in group to determine correct file name.
Data-base-access file name too long	File name contains more than six characters.	Rename file. Try DBUTIL command again.
DATA BASE ALREADY CREATED	User has invoked DBUTIL utility in CREATE mode and has specified the name of a data base which already exists.	In session mode, correct error or press *return* to terminate program.
DATA BASE IN USE	Utility program cannot gain exclusive access to the referenced data base because of other current users.	

Table A-8. IMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
Data base is remote	To use DBUTIL, you must be logged on to the same group and account containing the root file.	Do a remote logon and run DBUTIL from your remote session.
Data base name too long	Data base name specified contains more than six characters.	Try command again with correct name.
Data base or access file does not exist.	No such file exists in log-on group and account.	Check files in group to determine correct file name.
Data base or access file name required	Specified.command must include data base or access file name.	Reenter command.
Data base or access file name too long.	File name contains more than six characters.	Reenter command with correct name.
DATA BASE REQUIRES CREATION	Data base creator must run the DBUTIL program in CREATE mode prior to executing DBUNLOAD, DBLOAD, or DBUTIL in ERASE mode.	In session mode, correct error or press *return* to terminate program.
DUPLICATE FILE NAME	Required file name is already assigned to some other file. For example, if data base STORE requires six data sets (STORE01 – STORE06), then a previously defined file STORE03 would trigger this message. It can occur if data base is not purged before executing DBRESTOR.	
Error reading root file label	DBUTIL is unable to read the root file	Contact your HP Systems Engineer
Error writing root file label	DBUTIL has detected an error while writing the root file label.	Contact your HP Systems Engineer.
EXCEEDS ACCOUNT DISC SPACE	Amount of disc space required by data base plus that already assigned to other files of this account exceeds the amount of disc space available to the account.	Request system manager to increase account's disc space.
EXCEEDS GROUP DISC SPACE	Amount of disc space required by data base plus that already assigned to other files of this group exceeds the amount of disc space available to the account.	Request system manager to increase group's disc space.

Table A-8. IMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
FCHECK failure FCLOSE failure FCONTROL failure FENTRY failure FGETINFO failure	Exceptional errors indicating a hardware or software failure.	Notify system manager of error
File equates are illegal for data base and data-base-access files	DBUTIL does not allow you to equate the name of the data base or data-base-access file to another file name using the :FILE command.	Use the :RESET command to cancel the :FILE command. (Either break and resume execution or exit DBUTIL and run it again.)
FREAD error on ASCII access file FREADDIR failure	Exceptional errors indicating a hardware or software failure.	Notify system manager of error
INSUFFICIENT DISC SPACE	Amount of disc space required for data base is not available from the system.	Consult with system manager about disc space requirements.
INSUFFICIENT VIRTUAL MEMORY	Amount of virtual memory available is insufficient to open and access data base.	Try running utility later when system is not so busy.
Invalid contents of ASCII access file	Specified data-base-access file does not conform to required record formats.	Use the Editor to check file contents.
Invalid Data Base Control Block	IMAGE has encountered an inconsistency in the DBCB.	Contact your HP Systems Engineer.
Invalid data-base-access file name Invalid data base name Invalid data base name or access file name	File name or data base name must be 1 to 6 alphanumeric characters beginning with an alphabetic.	Reenter command with correct name.
Invalid delimiter	Comma or space incorrectly positioned.	Use HELP to check command syntax and reenter command.

Table A-8. IMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
Invalid maintenance word	–	Check maintenance word specified, reenter command.
Invalid number of buffers specified	The number of buffers must be > = 4 and < = 255.	Enter the SET command again.
Invalid number of users specified	The minimum number of users allowed is 1 and the maximum is 120. The ranges must be strictly increasing in order.	Enter the SET command again.
Invalid parameter	An incorrect parameter has been specified.	Use HELP to check command format and reenter command.
Maintenance word required	User invoking utility is not the creator of referenced data base and has failed to supply a maintenance word.	If DBUTIL, reenter command with maintenance word. Otherwise, correct error or press *return* to terminate program.

Table A-8. IMAGE Utility Program Conditional Messages (Continued)

Maintenance word too long	Maintenance word specified contains more than 8 characters.	Reenter command with correct maintenance word.
Non-creator access not permitted	You must be the data base creator to perform this function.	—
NO SUCH DATA BASE	Specified data base does not exist in user's log on group.	Check *base name* and log on account and group. Press *return* if want to terminate.
Not a data-base-access file	Specified file is not a data-base-access file.	Check file name and try command again.
Not a data base or data-base-access file	—	Check data base or file name and try command again.
Not a data base root file	—	Check specified data base name. Try command again.
Not a privileged data-base-access file	Data-base-access file has not been activated.	Check command and reenter correctly.
NOT ALLOWED; MUST BE CREATOR	User invoking utility is not the creator of the data base and data base has no maintenance word.	Log on with correct user name, account and group.
Not an unprivileged data-base-access file	Data-base-access file is already activated.	Check command and reenter command if desired.
OUTMODED ROOT	Root file of specified data base corresponds to different version of IMAGE software and is not compatible with utility program currently executing.	Consult with system manager to verify correct version of software. If necessary ask HP support personnel about conversion process.

Table A-8. IMAGE Utility Program Conditional Messages (Continued)

MESSAGE	MEANING	ACTION
Parameter expected	Specified command requires another parameter.	Use HELP to determine correct command syntax.
Parameter must be a command	Only command names may be specified with HELP command.	Reenter command.
Parameter specified twice	–	Reenter command.
Premature EOF on ascii access file	DBUTIL encountered an end-of-file mark before at least one =HELLO record in the data-base-access file.	Alter the file content using the Editor.
Root file does not exist	A root file with the name of the specified data base does not exist in the log-on group and account.	Check data base name specified and reenter command.
Too many parameters	–	Reenter command with correct number of parameters.
Unknown command, try HELP	DBUTIL does not recognize specified command.	Enter HELP command to get list of all DBUTIL commands.

Table A-9. IMAGE Utility Program Unconditional Messages

MESSAGE	MEANING	ACTION
AUTOMATIC MASTER IS FULL ON PATH NUMBER n	DBLOAD is unable to load a detail entry because automatic master data set associated with path n of detail data set is full.	Recreate root file with larger capacity for automatic master. Rerun necessary utilities.
BAD DATA BASE	This message is issued by DBSTORE, DBRESTOR, and DBUNLOAD. It means the data base is flagged "bad" because of a known structural error probably due to an abnormal termination or to a system crash during DBLOAD or some other "output deferred" operation. The current operation (DBSTORE, DBRESTOR, or DBUNLOAD) continues to function normally. DBLOAD additionally prints the message "SERIAL UNLOAD FOLLOWS", and automatically operates in serial mode. The data base on disc retains its "bad" flag, and cannot be accessed through DBOPEN.	The data base is not usable in its current state. Purge it and restore a backup copy, or erase it and then load it from a tape or serial disc written by DBUNLOAD or from some other external copy of the data.
Cannot open terminal, terminating	DBUTIL is unable to access the terminal.	Call HP Systems Engineer.
CHAIN IS FULL ON PATH NUMBER n	DBLOAD is unable to load a detail entry because the chain count for path number n of detail data set exceeds $2^{16}-1$ (or 65535 entries).	Either delete some entries from the chain and reload or change data base design if necessary.
DATA BASE UTILITY ERROR: $p_1/p_2/p_3/p_4/p_5$	This message occurs when an unusual error condition is returned by an IMAGE library procedure. p_1, p_2, p_3 are usually the first three words of status area returned by library procedure. Values of parameters depend upon procedure and specific error condition. p_4 is the PCODE of library procedure, a number uniquely assigned to the procedure to identify it. p_5 is an octal PB-relative return address within code segment of utility which initiated error message. (p_1 through p_4 are decimal while p_5 is octal.)	For more information about the meaning of the parameters, see description of status array following table A-3. Also see Section IV for a description of library procedures.
DATA SET FULL	Data set currently being loaded is full.	Recreate root file, increase data set's capacity. Run utilities again.

Table A-9. IMAGE Utility Program Unconditional Messages (Continued)

MESSAGE	MEANING	ACTION
DBSTORE FAILED – NO DATA BASE STORED	A file error or other system message follows explaining the problem.	Contact your HP Systems Engineer.
Device must be TAPE or SERIAL DISC	The output device for the DBSTORE file must be a magnetic tape or serial disc.	Run DBSTORE again and change :FILE command if necessary.
EOF seen, program terminating	A :EOD has been entered.	Run DBUTIL again.
File equate for { DBSTORE DBRESTOR DBLOAD DBUNLOAD } may only use DEV	If you specify an input/output file with a :FILE command for any of these utility programs, only the file designators and DEV= parameter are allowed.	Enter :FILE command again and rerun utility program.
Hard terminal read error, terminating	DBUTIL cannot read input from terminal.	Notify HP Systems Engineer.
INVALID SET COUNT	IMAGE has detected an inconsistency in the data set count.	Contact yoru HP Systems Engineer.
NO MANUAL ENTRY FOR DETAIL ON PATH NUMBER n	DBLOAD is attempting to load a detail data set entry. n is the number of the detail data set path referencing the manual master in question.	Add entry to manual master with application program or QUERY. RUN DBLOAD again.
UNABLE TO CONTINUE	DBUTIL program (operating in PURGE mode) cannot continue execution due to exceptional error in file system. This information is followed by MPE file information display. See Appendix A.	Save file information. Consult with system manager and HP support personnel if necessary.

RESULTS OF MULTIPLE ACCESS

APPENDIX B

When opening a data base with DBOPEN, IMAGE returns information in the *status* array describing the results of the procedure call. Table B-1 can be used to interpret these results when multiple processes are using the data base.

Each box in Table B-1 is associated with a requested mode or IMAGE utility routine identified at the far left of the row in which the box appears. It is also associated with a "possible" current access mode or utility routine identified at the top of the column in which the box appears. The contents of the boxes can be used to determine the results of a DBOPEN call.

If access is granted, condition code CCE is returned and the first word of the *status* array contains a zero. The boxes containing "G" represent this situation.

If access is not granted, and the reason relates to current data base activity, the results are like those shown in the other boxes. There are two types of situations:

- If the first two words of *status* contain -1 and 0 respectively, the third word of *status* will contain a single number.* If that number is 48, 90, or 91, the failure occurred because current access to the data base does not permit it to be opened in the requested mode. Find the boxes in the "requested mode" row which contain a number equal to the third *status* word. The possible modes and utility routines which other processes may be using are the ones which label the columns containing these boxes. For example, if the third *status* word contains 48 and the requested mode is 2, the possible current modes are 1 and 5.

  To find an alternate mode for accomplishing the task, look down the columns containing these boxes for one containing a "G". If the requested mode labeling the row in which the "G" resides can be used, try opening the data base with that mode. In the example above, alternate modes would be 1 or 5 since these rows contain "G" in columns 1 and 5.

  If the box with contents matching the third *status* word is in a column associated with a utility, usually the only choice is to wait until execution terminates. When DBSTORE is being run, it is possible to open the data base with mode 6 or 8.

- If the first word in the *status* array contains -32, the failure occurred because the root file could be opened but not with the necessary AOPTIONS.** Use the same technique described above to determine the possible current modes or other activity and to select a course of action. For example, if the requested mode is 2 and the first word of *status* equals -32, possible current modes are 4 and 8, and the DBSTORE utility may be executing.

Messages enclosed in quotes are printed when the situation represented by the row and column headings occurs.

*This number is the MPE failure code returned from the FCHECK intrinsic. See the *MPE Intrinsics Reference Manual* for MPE failure code meanings.

**This value can also be returned in situations not related to multiple access. See Appendix A in this manual and the description of the AOPTIONS parameter of the FOPEN intrinsic in the *MPE Intrinsics Reference Manual.*

Table B-1. Actions resulting from Multiple Access of Data Bases

		CURRENT DBOPEN MODE										
		1	2	3	4	5	6	7	8	BEING DBSTOREd	BEING DBRESTORed	BEING DBUNLOADed OR DBLOADed
REQUESTED DBOPEN MODE	1	G	48	91	48	G	48	91	48	48	91	91
	2	48	G	91	-32	48	G	91	-32	-32	91	91
	3	90	90	91	90	90	90	91	90	90	91	91
	4	90	90	91	90	48	G	91	-32	-32	91	91
	5	G	48	91	48	G	48	91	48	48	91	91
	6	48	G	91	G	48	G	91	G	G	91	91
	7	90	90	91	90	90	90	91	90	90	91	91
	8	90	90	91	90	48	G	91	G	G	91	91
:RUN DBSTORE		"DATA BASE IN USE"					G	"DATA BASE IN USE"	G	G	"DATA BASE IN USE"	
:RUN DBRESTOR		"DUPLICATE FILE NAME"										
:RUN DBUNLOAD OR :RUN DBLOAD		"DATA BASE IN USE"										

Note: 48, 90, and 91 are values returned in the **third** *status* word and −32 is a value returned in the **first** *status word.* G indicates access is granted.

SUMMARY OF DESIGN CONSIDERATIONS

APPENDIX C

1. Keep one-of-a-kind information in master data sets, such as, unique identifiers. Keep duplicate information in detail data sets, such as, records of events (sales, purchases, shipments).

2. Define a search item in a detail data set if you want to retrieve all entries with a common value for that data item.

3. Use manual master data sets to prevent entry of invalid data in the detail search item linked to the master through a path. Use automatic master data sets to save time if the detail search items are unpredictable or too numerous to enter manually.

4. Limit the use of sort items to paths with relatively short chains in order to reduce the time required to add and delete entries.

5. Select the path most frequently accessed in chained order as the primary path.

6. Remember that data items must be an integral number of words in length.

7. When selecting the maximum block size, consider the environment in which the data base will be used. (Refer to Section III, $CONTROL command for more information.)

8. If you intend to use QUERY with your data base, use data types that QUERY accepts and avoid using compound items, if possible.

9. In application programs either reference data items and data sets by name or use DBINFO at the beginning of the program to initialize the data item and data set numbers in order to maintain data independence of the programs.

   Refer to the discussion in Section IV to decide on appropriate access modes to use for your application programs.

10. Analyze the time required to maintain the data base, for example, the time required to unload and load the data base. The HP 3000 Contributed Library package IDEA* can assist you in performing this analysis. It can also assist you in measuring the type of performance you will get with the application based on the number of sessions running concurrently and the amount of activity against the data base.

11. The capacity of each data set should be defined as realistically as possible since a capacity that is too large wastes disc space. The capacity can be increased when necessary by restructuring the data base as described in Section VI.

12. A master data set capacity equal to a prime number or to the product of two or three primes generally yields fewer synonyms than a master data set capacity of many prime factors.

13. The account and group in which the data base resides must have enough file space available to contain all the data base files.

*This package is offered by HP General Systems Division. Contact your local HP Sales Office for more information.

14. If your application uses sorted paths, plan to add or delete entries (DBPUT,DBDELETE) to sorted chains when the system is not very busy. If it is very busy, limit the data base activity on sorted chains to reading and updating (DBUPDATE).

15. Do most of your locking at one level (data base, data set, or data entry).

16. If locking at the data entry level, do most of the locking using one item in each data set. Otherwise, performance will be the same as if you were locking at the data set level.

MULTIPLE RIN SPECIAL CAPABILITY

APPENDIX

D

For the purpose of deadlock prevention, the system views any call to DBLOCK in which something is actually locked as a lock on a single resource, even though the call may have specified multiple lock descriptors. Any program which does not have the Multiple RIN capability (CAP=MR) can only have one resource locked at a time, and thus can only call DBLOCK once without an intervening call to DBUNLOCK.

It may be necessary for some applications to violate this rule. The purpose of this appendix is to tell you how to avoid problems that may arise if you prepare your application programs with MR capability (CAP=MR).

Some typical situations in which CAP=MR may be required are the following:

- A program has two or more data bases open and wishes to lock part or all of each data base simultaneously. (One or more of the data bases may be on a remote HP 3000.)

- A program wishes to lock an MPE file and a data base simultaneously.

- A program wishes to lock data entries in a data base and, after reading their contents, to apply further locks. This is very dangerous and is not recommended, since deadlocks can occur very easily.

The danger in all cases is that a deadlock may occur. For example, suppose process A has data set 1 locked and is trying to lock data set 9, and process B has data set 9 locked and is waiting for data set 1. In this case, a deadlock has occurred and the only way to break it is to restart the operating system.

IMAGE avoids deadlocks within single calls to DBLOCK by first sorting the lock descriptors into an internally-defined sequence. It then applies the locks in ascending sorted order. You can use the same strategy in avoiding deadlocks. First define an order in which entities should be locked and then impose a rule on all programmers that this order be adhered to. The sequence of unlocking is not important. The rule that you establish should apply to all lockable entities:

- data bases, data sets, and data entries

- remote data bases, data sets, and data entries

- MPE files (FLOCK), global RINs (LOCKGLORIN), KSAM files (FLOCK), and files locked with the COBOLLOCK procedure.

When applying multiple DBLOCK calls to the same data base, extreme caution should be exercised since the deadlock situations can be very subtle. For example, if a process locks a data set and then attempts to lock the data base, the process will wait for itself forever.

If it is absolutely necessary to make multiple DBLOCK calls, the following information about how IMAGE performs locking may be useful.

SORT SEQUENCE FOR LOCK DESCRIPTORS

IMAGE internally sorts the lock descriptors in ascending order as follows:

- Major key — data set number
- Minor key — lower bound of data item value

If a lock descriptor's *relop* field contains < = , it collates before any other lock descriptors for the data set since it has the lowest possible lower bound for it's value. For example, a lock descriptor of SALES:QUANTITY < = 10 collates before a lock descriptor of SALES:QUANTITY = 5, since the lower bound of the former is the lowest possible integer value for an I-type data item.

CONDITIONAL LOCKS

During a DBLOCK, if IMAGE discovers a lock descriptor that is identical to one previously put into effect by the same user through the same access path, it ignores the latest lock descriptor. For example, the lock descriptor SALES: ACCOUNT = 89393899 is ignored if SALES: ACCOUNT = 89393899 was locked earlier on the same access path. However, it will not be ignored if a lock descriptor such as SALES: @ has been specified earlier.

If multiple lock descriptors are specified with mode 6 (conditional data entry locking), IMAGE indicates how many locks have been applied when it returns to the calling process. It does not release the successful locks even though all the requested locks have not been applied. Since IMAGE ignores identical lock descriptors specified a second time, it is possible to call DBLOCK again with the same descriptor list (if the program has MR capability). Those lock descriptors that are already in effect will be ignored and the others will be tried again. The second word of the *status* array contains the number of descriptors successfully locked in each call. This technique will not cause deadlocks provided the lock descriptor list is not altered.

It is not recommeded that this technique be used in a tight program loop since system performance will degarde markedly. However, it can be used to retry the locks in a situation where the program prompts the user to determine whether the locks should be tried again, and the user indicates that they should. (If the user does not want to continue trying to lock all the entities, be sure to unlock the ones that succeeded.)

REMOTE DATA BASES

Locking remote data base entities is the same as locking local data base entities with the following exception. If the local system has a user-created process structure, and each process is locking a remote data base independently, the programs must have Multiple RIN capability since they are in the same job/session. The only effect using MR capability has on the local system is that the rule prohibiting multiple DBLOCK calls is not enforced. However, to access remote data bases each local process must issue a separate REMOTE HELLO to ensure that it has a corresponding (sister) process in the remote system.

The system does not force you to establish corresponding remote processes, but failure to do so can result in the remote session permanently suspending and requires a remote system restart to recover.

> Hewlett-Packard does not accept responsibility for system lockouts and deadlocks when Multiple RIN capability is in use with IMAGE/3000.

The DBUTIL command SHOW *data-base-name* LOCKS may be useful in tracing deadlocks that occur when CAP=MR is used.

INDEX

READER COMMENT SHEET

HP 3000 Computer System
IMAGE
Reference Manual

32215-90003 Sep 1978

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate? Yes □ No □ (If no, explain under Comments, below.)

Are the concepts and wording easy to understand? Yes □ No □ (If no, explain under Comments, below.)

Is the format of this manual convenient in size, arrangement, and readability? Yes □ No □ (If no, explain or suggest improvements under Comments, below.)

Comments:

FROM:

Name ____________________

Company ____________________

Address ____________________

____

____

FOLD FOLD

FIRST CLASS
PERMIT NO. 1020
SANTA CLARA
CALIFORNIA

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States. Postage will be paid by

Publications Manager
Hewlett-Packard Company
General Systems Division
5303 Stevens Creek Boulevard
Santa Clara, California 95050

FOLD FOLD

HEWLETT PACKARD

SALES & SERVICE OFFICES

AFRICA, ASIA, AUSTRALIA

ANGOLA
Telectra
Empresa Técnica de
Equipamentos
Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 42-I°DT.°
Caixa Postal, 6487
Luanda
Tel: 35515/6
Cable: TELECTRA Luanda

AUSTRALIA
Hewlett-Packard Australia
Pty. Ltd.
31-41 Joseph Street
Blackburn, Victoria 3130
P.O. Box 36
Doncaster East, Victoria 3109
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne

Hewlett-Packard Australia
Pty. Ltd.
31 Bridge Street
Pymble
New South Wales, 2073
Tel: 449-6566
Telex: 21561
Cable: HEWPARD Sydney

Hewlett-Packard Australia
Pty. Ltd.
153 Greenhill Road
Parkside, S.A., 5063
Tel: 272-5911
Telex: 82536
Cable: HEWPARD Adelaide

Hewlett-Packard Australia
Pty.Ltd.
141 Stirling Highway
Nedlands, W.A. 6009
Tel: 86-5455
Telex: 93859
Cable: HEWPARD Perth

Hewlett-Packard Australia
Pty. Ltd.
121 Wollongong Street
Fyshwick, A.C.T. 2609
Tel: 95-2733
Telex: 62650
Cable: HEWPARD Canberra

Hewlett Packard Australia
Pty. Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
Spring Hill, 4000 Queensland
Tel: 229-1544
Cable: HEWPARD Brisbane

GUAM
Medical/Personal Calculators Only
Guam Medical Supply, Inc.
Jay Ease Building, Room 210
P.O. Box 8947
Tamuning 96911
Tel: 646-4513
Cable: EARMED Guam

HONG KONG
Schmidt & Co.(Hong Kong) Ltd.
P.O. Box 297
Connaught Centre,
39th Floor
Connaught Road, Central
Hong Kong
Tel: H-255291-5
Telex: 74766 SCHMC HX
Cable: SCHMIDTCO Hong Kong

INDIA
Blue Star Ltd.
Kasturi Buildings
Jamshedji Tata Rd
Bombay 400 020
Tel: 29 50 21
Telex: 011-2156
Cable: BLUEFROST

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
Bombay 400 025
Tel: 45 78 87
Telex: 011-4093
Cable: FROSTBLUE

Blue Star Ltd.
Band Box House
Prabhadevi
Bombay 400 025
Tel: 45 73 01
Telex: 011-3751
Cable: BLUESTAR

Blue Star Ltd.
7 Hare Street
P.O. Box 506
Calcutta 700 001
Tel: 23-0131
Telex: 021-7655
Cable: BLUESTAR

Blue Star Ltd.
Bhandari House
7th & 8th Floor
91 Nehru Place
New Delhi 110 024
Tel: 634770 & 635166
Telex: 031-2463
Cable: BLUESTAR

Blue Star Ltd.
Blue Star House
11/11A Magarath Road
Bangalore 560 025
Tel: 55668
Telex: 043-430
Cable: BLUESTAR

Blue Star Ltd.
Meeakshi Mandiram
xxx/1678 Mahatma Gandhi Rd.
Cochin 682 016
Tel: 32069,32161,32282
Telex: 0885-514
Cable: BLUESTAR

Blue Star Ltd.
1-1-117/1
Sarojini Devi Road
Secunderabad 500 003
Tel: 70126, 70127
Telex: 015-459
Cable: BLUEFROST

Blue Star Ltd.
2/34 Kodambakkam High Road
Madras 600 034
Tel: 82056
Telex: 041-379
Cable: BLUESTAR

INDONESIA
BERCA Indonesia P.T.
P.O. Box 496/Jkt.
Jln.Abdul Muis 62
Jakarta
Tel: 40369, 49886,49255,356038
Telex: Jkt.42895
Cable: BERCACON

BERCA Indonesia P.T.
P.O. Box 174/Sby.
23 Jln. Jimerto
Surabaya
Tel: 42027
Cable: BErcacon

ISRAEL
Electronics Engineering Div.
of Motorola Israel Ltd.
16, Kremenetski Street
P.O.Box 25016
Tel-Aviv
Tel: 38973
Telex: 33569
Cable: BASTEL Tel-Aviv

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4th Floor
4-20, Nishinakajima 5-chome
Yodogawa-ku, Osaka-shi
Osaka,532
Tel: 06-304-6021
Telex: 523-3624

Yokogawa-Hewlett-Packard Ltd.
29-21, Takaido-Higashi 3-chome
Suginami-ku, **Tokyo** 168
Tel: 03-331-6111
Telex: 232-2024 YHP-Tokyo
Cable: YHPMARKET TOK 23 724

Yokogawa-Hewlett-Packard Ltd.
Nakamo Building
24 Kami Sasajima-cho
Nakamura-ku, **Nagoya**, 450
Tel: 052 571-5171

Yokogawa-Hewlett-Packard Ltd.
Tanigawa Building
2-24-1 Tsuruya-cho
Kanagawa-ku
Yokohama, 221
Tel: 045-312-1252
Telex: 382-3204 YHP YOK

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
105, 1-chome, San-no-maru
Mito, Ibaragi 310
Tel: 0292-25-7470

Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho, 1-chome
Atsugi, Kanagawa 243
Tel: 0462-24-0452

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi
Hachijuni Building
4th Floor
3-4, Tsukuba
Kumagaya, Saitama 360
Tel: 0485-24-6563

KENYA
Technical Engineering
Services(E.A.)Ltd.
P.O. Box 18311
Nairobi
Tel: 55679/556680/557726
Telex: 22629
Cable: PROTON

Medical Only
International Aeradio(E.A.)Ltd.
P.O. Box 19012
Nairobi Airport
Nairobi
Tel: 336055/56
Telex: 22201/22301
Cable: INTAERIO Nairobi

KOREA
Samsung Electronics Co., Ltd.
15th Floor, Daeyongak Bldg.,
25-5, 1-KA
Choong Moo-Ro, Chung-Ku,
Seoul
Tel: (23) 6811, 778-3401/2/3/4
Telex: 2257S
Cable: ELEKSTAR Seoul

MALAYSIA
Teknik Mutu Sdn. Bhd.
No. 2, Lorong 13/6A
Section 13
Petaling Jaya,**Selangor**
Tel: 54994/54916
Telex: MA 37605

Protel Engineering
P.O. Box 1917
Lot 259, Satok Road
Kuching, **Sarawak**
Tel: 53544
Cable: PROTELENG

MOZAMBIQUE
A.N. Goncalves, Ltd.
162, 1° Apt. 14 Av. D. Luis
Caixa Postal 107
Maputo
Tel: 27091, 27114
Telex: 6-203 NEGON Mo
Cable: NEGON

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
Kilbirnie, Wellington 3
P.O. Box 9443
Courtney Place
Wellington
Tel: 877-199
Cable: HEWPACK Wellington

Hewlett-Packard (N.Z.) Ltd.
Pakuranga Professional Centre
267 Pakuranga Highway
Box 51092
Pakuranga
Tel: 569-651
Cable: HEWPACK Auckland

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
Scientific Division
79 Carlton Gore Road, Newmarket
P.O. Box 1234
Auckland
Tel: 75-289
Cable: DENTAL Auckland

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
Private Bag
Norrie and Parumoana Streets
Porirua
Tel: 75-098
Telex: 3858

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
P.O. Box 309
239 Stanmore Road
Christchurch
Tel: 892-019
Cable: DENTAL Christchurch

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
303 Great King Street
P.O. Box 233
Dunedin
Tel: 88-817
Cable: DENTAL Dunedin

NIGERIA
The Electronics
Instrumentations Ltd.
N6B/770 Oyo Road
Oluseun House
P.M.B. 5402
Ibadan
Tel: 61577
Telex: 31231 TEIL Nigeria
Cable: THETEIL Ibadan

The Electronics Instrumentations Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
Lagos
Cable: THETEIL Lagos

PAKISTAN
Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
Karachi-3
Tel: 511027, 512927
Telex: 2894
Cable: COOPERATOR Karachi

Mushko & Company, Ltd.
38B, Satellite Town
Rawalpindi
Tel: 41924
Cable: FEMUS Rawalpindi

PHILIPPINES
The Online Advanced
Systems Corporation
Rico House
Amorsolo cor. Herrera Str.
Legaspi Village, Makati
P.O. Box 1510
Metro **Manila**
Tel: 85-35-81, 85-34-91,85-32-21
Telex: 3274 ONLINE

RHODESIA
Field Technical Sales
45 Kelvin Road North
P.O. Box 3458
Salisbury
Tel: 705231 (5 lines)
Telex: RH 4122

SINGAPORE
Hewlett-Packard Singapore
(Pte.) Ltd.
1150 Depot Road
Alexandra P.O. Box 58
Singapore 4
Tel: 270-2355
Telex: HPSG RS 21486
Cable: HEWPACK, Singapore

SOUTH AFRICA
Hewlett-Packard South Africa
(Pty.), Ltd.
Private Bag Wendywood,
Sandton, Transvaal, 2144
Hewlett-Packard Centre
Daphne Street, Wendywood,
Sandton, 2144
Tel: 802-1040/8
Telex: 8-4782
Cable: HEWPACK Johannesburg

Hewlett-Packard South Africa
(Pty.), Ltd.
P.O. Box 120
Howard Place, Cape Province, 7450
Pine Park Centre, Forest Drive,
Pinelands, Cape Province, 7405
Tel: 53-7955 thu 9
Telex: 57-0006

TAIWAN
Hewlett-Packard Far East Ltd.
Taiwan Branch
39 Chung Hsiao West Road
Section 1, 7th Floor
Taipei
Tel: 3819160-4,3141010,3715121
Ext. 270-279
Cable: HEWPACK TAIPEI

Hewlett-Packard Far East Ltd.
Taiwan Branch
68-2, Chung Cheng 3rd. Road
Kaohsiung
Tel: (07) 242318-Kaohsiung

Analytical Only
San Kwang Instruments Co., Ltd.
20 Yung Sui Road
Taipei
Tel: 3715171-4 (5 lines)
Telex: 22894 SANKWANG
Cable: SANKWANG Taipei

TANZANIA
Medical Only
International Aeradio (E.A.), Ltd.
P.O. Box 861
Dar es Salaam
Tel: 21251 Ext. 265
Telex: 41030

THAILAND
UNIMESA Co. Ltd.
Elcom Research Building
2538 Sukumvit Ave.
Bangchak,Bangkok
Tel: 3932387, 3930338
Cable: UNIMESA Bangkok

UGANDA
Medical Only
International Aeradio(E.A.), Ltd.
P.O. Box 2577
Kampala
Tel: 54388
Cable: INTAERIO Kampala

ZAMBIA
R.J. Tilbury (Zambia) Ltd.
P.O. Box 2792
Lusaka
Tel: 73793
Cable: ARJAYTEE, Lusaka

OTHER AREAS NOT LISTED, CONTACT:
Hewlett-Packard Intercontinental
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 856-1501
TWX: 910-373-1267
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

CANADA

ALBERTA
Hewlett-Packard (Canada) Ltd.
11620A - 168th Street
Edmonton T5M 3T9
Tel: (403) 452-3670
TWX: 610-831-2431

Hewlett-Packard (Canada) Ltd.
210,7220 Fisher St. S.E.
Calgary T2H 2H8
Tel: (403) 253-2713
Twx: 610-821-6141

BRITISH COLUMBIA
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
Richmond V6X 2W7
Tel: (604) 270-2277
TWX: 610-925-5059

MANITOBA
Hewlett-Packard (Canada) Ltd.
380-550 Century St.
Winnipeg R3H 0Y1
Tel: (204) 786-6701
TWX: 610-671-3531

NOVA SCOTIA
Hewlett-Packard (Canada) Ltd.
800 Windmill Road
Dartmouth B3B 1L1
Tel: (902) 469-7820
TWX: 610-271-4482 HFX

ONTARIO
Hewlett-Packard (Canada) Ltd.
1020 Morrison Dr.
Ottawa K2H 8K7
Tel: (613) 820-6483
TWX: 610-563-1636

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
Mississauga L4V 1M8
Tel: (416) 678-9430
TWX: 610-492-4246

QUEBEC
Hewlett-Packard (Canada) Ltd.
275 Hymus Blvd.
Pointe Claire H9R 1G7
Tel: (514) 697-4232
TWX: 610-422-3022
TLX: 05-821521 HPCL

FOR CANADIAN AREAS NOT LISTED:
Contact Hewlett-Packard (Canada)
Ltd. in Mississauga.

CENTRAL AND SOUTH AMERICA

ARGENTINA
Hewlett-Packard Argentina
S.A.
Av. Leandro N. Alem 822 - 12°
1001 **Buenos Aires**
Tel: 31-6063,4,5,6
Telex: 122443 AR CIGY
Cable: HEWPACKARG

Biotron S.A.C.I.y M.
Bolivar 177
1066 **Buenos Aires**
Tel: 30-4846, 34-9356, 34-0460,
Telex: 17595

BOLIVIA
Casa Kavlin S.A.
Calle Potosi' 1130
P.O. Box 500
La Paz
Tel: 41530,53221
Telex: CWC BX 5298,ITT 3560082
Cable: KAVLIN

BRAZIL
Hewlett-Packard do Brasil
I.e.C. Ltda.
Alameda Rio Negro, 750
Alphaville
06400 **Barueri** SP
Tel: 429-3222
Cable: HEWPACK Sao Paulo

Hewlett-Packard do Brasil
I.e.C. Ltda.
Rua Padre Chagas, 32
90000-**Pórto Alegre**-RS
Tel: (0512) 22-2998, 22-5621
Cable: HEWPACK Porto Alegre

Hewlett-Packard do Brasil
I.e.C. Ltda.
Rua Siqueira Campos, 53
Copacabana
20000-**Rio de Janeiro**-RJ
Tel: 257-80-94-DDD (021)
Telex: 391-212-1905 HEWP-BR
Cable: HEWPACK
Rio de Janeiro

CHILE
Calcagni y Metcalfe Ltda.
Alameda 580-Of. 807
Casilla 2118
Santiago, 1
Tel: 398613
Telex: 3520001 CALMET
Cable: CALMET Santiago

COLOMBIA
Instrumentación
Henrik A. Langebaek & Kier S.A.
Carrera 7 No. 48-75
Apartado Aéreo 6287
Bogotá, I D.E.
Tel: 69-88-77
Telex: 044-400
Cable: AARIS Bogotá

Instrumentacion
H.A. Langebaek & Kier S.A.
Apartado Aereo 54098
Medellin
Tel: 304475

COSTA RICA
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
San Jose
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR CR
Cable: GALGUR

ECUADOR
Computadoras y Equipos
Electrónicos
P.O. Box 6423 CCI
Eloy Alfaro No. 1824,3°Piso
Quito
Tel: 453 482
Telex: 2548 CYEDE ED
Cable: Sagita-Quito

Medical Only
Hospitalar S.A.
Casilla 3590
Robles 625
Quito
Tel: 545-250
Cable: Hospitalar-Quito

EL SALVADOR
Instrumentacion y Procesamiento
Electronico de el Salvador
Bulevar de los Heroes 11-48
San Salvador
Tel: 252787

GUATEMALA
IPESA
Avenida Reforma 3-48,
Zona 9
Guatemala City
Tel: 316627,314786,66471-5,ext.9
Telex: 4192 Teletro Gu

MEXICO
Hewlett-Packard Mexicana,
S.A. de C.V.
Av. Periférico Sur No. 6501
Tepepan, Xochimilco
Mexico 23, D.F.
Tel: 905-676-4600
Telex: 017-74-507

Hewlett-Packard Mexicana,
S.A. de C.V.
Ave. Constitución No. 2184
Monterrey, N.L.
Tel: 48-71-32, 48-71-84
Telex: 038-410

NICARAGUA
Roberto Terán G.
Apartado Postal 689
Edificio Terán
Managua
Tel: 25114, 23412,23454,22400
Cable: ROTERAN Managua

PANAMA
Electrónico Balboa, S.A.
P.O. Box 4929
Calle Samuel Lewis
Culdad de Panama
Tel: 64-2700
Telex: 3485126 Curundu,
Canal Zone
Cable: ELECTRON Panama

PERU
Compañla Electro Médica S.A.
Los Flamencos 145
San Isidro Casilla 1030
Lima 1
Tel: 41-4325
Telex: Pub. Booth 25424 SISIDRO
Cable: ELMED Lima

URUGUAY
Pablo Ferrando S.A.
Comercial e Industrial
Avenida Italia 2877
Casilla de Correo 370
Montevideo
Tel: 40-3102
Telex: 702 PUBLIC BOOTH PARA
PABLO FERRANDO TELEFONO
40-31-02
Cable: RADIUM Montevideo

VENEZUELA
Hewlett-Packard de Venezuela
C.A.
P.O. Box 50933
Caracas 105
Los Ruices Norte
3a Transversal
Edificio Segre
Caracas 107
Tel: 35-00-11 (20 lines)
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas

FOR AREAS NOT LISTED, CONTACT:
Hewlett-Packard
Inter-Americas
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 856-1501
TWX: 910-373-1260
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

8/78

EUROPE, NORTH AFRICA AND MIDDLE EAST

AUSTRIA
Hewlett-Packard Ges.m.b.H.
Handelskai 52
P.O. Box 7
A-1205 **Vienna**
Tel: 351620-29
Cable: HEWPAK Vienna
Telex: 75923 hewpak a

BELGIUM
Hewlett-Packard Benelux
S.A./N.V.
Avenue du Col-Vert, 1,
(Groenkraaglaan)
B-1170 **Brussels**
Tel: (02) 660 0047,672-2240
Cable: PALOBEN Brussels
Telex: 23-494 paloben bru

CYPRUS
Kypronics
19 Gregorios Xenopoulos Street
P.O. Box 1152
Nicosia
Tel: 45628/29
Cable: Kypronics Pandehis
Telex: 3018

CZECHOSLOVAKIA
Vyvojova a Provozni Zakladna
Vyzkumnych Ustavu v Bechovicich
CSSR-25097 **Bechovice u Prahy**
Tel: 89 93 41
Telex: 121333

Institute of Medical Bionics
Vyskumny Ustav Lekarskej Bioniky
Jedlova 6
CS-88346
Bratislava-Kramare
Tel: 4251
Telex: 93229

DDR
Entwicklungslabor der TU Dresden
Forschungsinstitut Meinsberg
DDR-7305
Waldheim/Meinsberg
Tel: 37 667
Telex: 518741

Export Contact AG Zuerich
Guenther Forgber
Schlegelstrasse 15
1040 **Berlin**
Tel: 42-74-12
Telex: 111889

DENMARK
Hewlett-Packard A/S
Datavej 52
DK-3460 **Birkerod**
Tel: (02) 81 66 40
Cable: HEWPACK AS
Telex: 37409 hpas dk

Hewlett-Packard A/S
Navervej 1
DK-8600 **Silkeborg**
Tel: (06) 82 71 66
Telex: 37409 hpas dk
Cable: HEWPACK AS

EGYPT
I.E.A.
International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
Cairo
Tel: 23 829
Telex: 2067
Cable: INTENGASSO

Mohamed Sami Amin
Sami Amin Trading Office
18 Abdel Aziz Gawish
Abdine-Cairo
Tel: 24932
Cable: SAMITRO CAIRO

FINLAND
Hewlett-Packard OY
Nahkahousunti 5
P.O. Box 6
SF-00211 **Helsinki** 21
Tel: (90) 6923031
Cable: HEWPACKOY Helsinki
Telex: 12-1563 HEWPA SF

FRANCE
Hewlett-Packard France
Quartier de Courtaboeuf
Boite Postale No. 6
F-91401 **Orsay** Cedex
Tel: (1) 907 78 25
Cable: HEWPACK Orsay
Telex: 600048

Hewlett-Packard France
Bureau de vente de Lyon
"Le Saquin"
Chemin des Mouilles
B.P. 162
F-69130 **Ecully** Cedex
Tel: (78) 33 81 25,
Cable: HEWPACK Ecuiy
Telex: 31 06 17

Hewlett-Packard France
Bureau de vente de Toulouse
Péricentre de la Cépière
Chemin de la Cépière, 20
F-31300 **Toulouse-Le Mirail**
Tel:(61) 40 11 12
Cable: HEWPACK 51957
Telex: 510957

Hewlett-Packard France
Le Ligoures
Bureau de vente de Marseilles
Place Rouée de Villenueve
F-I3100 Aix-en-**Provence**
Tel: (42) 59 41 02
Cable: HEWPACK MARGN
Telex: 410770

Hewlett-Packard France
Bureau de vente de Rennes
2, Allee de la Bourgnette
B.P. 1124
F-35100 **Rennes** Cédex
Tel: (99) 51 42 44
Cable: HEWPACK 74912
Telex: 740912

Hewlett-Packard France
Bureau de vente de Strassbourg
I8, rue du Canal de la Marne
F-67300 **Schiltigheim**
Tel: (88) 83.08.10/83.11.53
Telex: 890141
Cable: HEWPACK STRBG

Hewlett-Packard France
Bureau de vente de Lille
Immeuble Péricentre
Rue van Gogh
F-59650 **Villeneuve** d Ascq
Tel: (20) 91.41.25
Telex: 16.01.24F

Hewlett-Packard France
Bureau de Vente
Centre d' affaires Paris-Nord
Bâtiment Ampère
Rue de la Commune de Paris
B.P. 300
F-93153 **Le Blanc Mesnil Cédex**
Tel: (01) 931 88 50

Hewlett-Packard France
Bureau de vente de Bordeau
Av. du Pdt. Kennedy
F-33700 **Meriguac**
Tel: (56) 97 22 69

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 **Frankfurt** 56
Tel: (0611) 50-04-1
Cable: HEWPACKSA Frankfurt
Telex: 04 13249 hpffm d

Hewlett-Packard GmbH
Technisches Büro Böblingen
Herrenberger Strasse 110
D-7030 **Böblingen**, Württemberg
Tel: (0703) 667-1
Cable: HEWPACK Böblingen
Telex: 07265739 bbn

Hewlett-Packard GmbH
Technisches Büro Düsseldorf
Emanuel-Leutze-Str.1(Seestern)
D-4000 **Düsseldorf**
Tel: (0211) 59711
Telex: 085/86 533 hpdd d

Hewlett-Packard GmbH
Technisches Büro Hamburg
Wendenstrasse 23
D-2000 **Hamburg** 1
Tel: (040) 24 13 93
Cable: HEWPACKSA Hamburg
Telex: 21 63 032 hphh d

Hewlett-Packard GmbH
Technisches Büro Hannover
Am Grossmarkt 6
D-3000 **Hannover 91**
Tel: (0511) 46 60 01
Telex: 092 3259

Hewlett-Packard GmbH
Technisches Büro Nürnberg
Neumeyerstrasse 90
D-8500**Nürnberg**
Tel: (0911) 56 30 83
Telex: 0623 860

Hewlett-Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 **Taufkirchen**
Tel: (089) 6117-1

Hewlett-Packard GmbH
Technisches Büro Berlin
Kaithstrasse 2-4
D-1000 **Berlin** 30
Tel: (030) 24 90 86
Telex:018 3405 hpbln d

GREECE
Kostas Karayannis
8 Omirou Street
Athens 133
Tel: 32 30 303/32/37 731
Telex: 21 59 62 RKAR GR
Cable: RAKAR ATHENS

Analytical Only
INTECO
G. Papathanassiou & Co.
17 Marni Street
Athens 103
Tel: 5522 915/5221 989
Telex: 21 5329 INTE GR
Cable: INTEKNIKA

Medical Only
Technomed Hellas Ltd.
52 Skoufa Street
Athens 135
Tel: 3626 972
Telex: 21 4693
Cable: ETALAK

HUNGARY
MTA
Müszerügy és Méréstechnikai
Szolgalata
Hewlett-Packard Service
Lenin Krt. 67, P.O.Box 241
1391Budapest VI
Tel: 42 03 38
Telex: 22 51 14

ICELAND
Medical Only
Elding Trading Company Inc.
Hafnarnvoli - Tryggvagötu
P.O.Box 895
IS-**Reykjavik**
Tel: 1 58 20/1 63 03
Cable: ELDING Reykjavik

IRAN
Hewlett-Packard Iran Ltd.
No. 13, Fourteenth St.
Mir Emad Avenue
P.O. Box 41/2419
Tehran
Tel: 851082-5
Telex: 213405 hewp ir

IRELAND
Hewlett-Packard Ltd.
King Street Lane
GB-**Winnersh,** Wokingham
Berks, RG11 5AR
Tel: (0734) 78 47 74
Telex: 847178
Cable: Hewpie London

ITALY
Hewlett-Packard Italiana S.p.A.
Via Amerigo Vespucci 2
Casella postale 3645
I-20124 **Milano**
Tel: (02) 6251 (10 lines)
Cable: HEWPACKIT Milano
Telex: 32046

Hewlett-Packard Italiana S.p.A.
Via Pellizzo 9
1-35100 **Padova**
Tel: (049) 66 48 88
Telex: 41612 Hewpacki

Hewlett-Packard Italiana S.p.A.
Via G. Armellini 10
1-00143 **Roma**
Tel: (06) 54 69 61
Telex: 61514
Cable: HEWPACKIT Roma

Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-10133 **Torino**
Tel:(011) 682245/659308

Medical/Calculators Only
Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43 G/C
I-95126 **Catania**
Tel:(095) 37 05 04

Hewlett-Packard Italiana S.p.A.
Via Amerigo Vespucci, 9
I-80142 **Napoli**
Tel: (081) 33 77 11
Telex: 61.51.4 Via Rome

Hewlett-Packard Italiana S.p.A.
Via E. Masi, 9/B
I-40137 **Bologna**
Tel: (5l) 30 78 87

JORDAN
Mouasher Cousins Co.
P.O. Box 1387
Amman
Tel: 24907/39907
Telex: SABCO JO 1456
Cable: MOUASHERCO

KUWAIT
Al-Khaldiya Trading &
Contracting
P.O. Box 830-Safat
Kuwait
Tel:42 4910/41 1726
Telex: 2481 Areeg kt
Cable: VISCOUNT

LUXEMBURG
Hewlett-Packard Benelux
S.A./N.V.
Avenue du Col-Vert, 1
(Groenkraaglaan)
B-1170 **Brussels**
Tel: (02) 672 22 40
Cable: PALOBEN Brussels
Telex: 23 494

MOROCCO
Dolbeau
81 rue Karatchi
Casablanca
Tel: 22 41 82/87
Telex: 23051/22833
Cable: MATERIO

Gerep
190 Blvd. Brahim Roudani
Casablanca
Tel: 25 16 76/25 90 99
Telex: 23 739
Cable: GEREP-CASA

Cogedir
2 Rue d' Agadir, B.P. 156
Casablanca
Tel: 27 65 40
Telex: 21 737
Cable: COGEDIR

NETHERLANDS
Hewlett-Packard Benelux N.V.
Van Heuven Goedhartlaan 121
P.O. Box 667
NL-Amstelveen 1134
Tel: (020) 47 20 21
Cable: PALOBEN Amsterdam
Telex: 13 216 hepa nl

NORWAY
Hewlett-Packard A/S
Osterdalen 18
P.O. Box 34
N-1345 Osteraas
Tel: (02) 1711 80
Telex: 16621 hpnas n

POLAND
Biuro Informacji Technicznej
Hewlett-Packard
Ul Stawki 2, 6P
00-950 **Warszawa**
Tel: 33.25.88/39.67.43
Telex: 81 24 53 hepa pl

UNIPAN
Biuro Obslugi Technicznej
01-447 Warszawa
ul **Newelska** 6
Poland

Zaklady Naprawcze Sprzetu
Medycznego
Plac Komuny Paryskiej 6
90-007 **Lódź**
Tel: 334-41, 337-83
Telex: 886981

PORTUGAL
Telectra-Empresa Técnica de
Equipamentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-**Lisbon** 1
Tel: (19) 68 60 72
Cable: TELECTRA Lisbon
Telex: 12598

Medical only
Mundinter
Intercambio Mundial de Comércio
S.a.r.l.
P.O. Box 2761
Avenida Antonio Augusto
de Aguiar 138
P - **Lisbon**
Tel: (19) 53 21 31/7
Telex: 16691 munter p
Cable: INTERCAMBIO Lisbon

RUMANIA
Hewlett-Packard Reprezentanta
Bd.n. Balcescu 16
Bucuresti
Tel: l5 80 23/13 88 85
Telex: 10440

I.I.R.U.C.
Intreprinderea Pentru
Intretinerea
Si Repararea Utilajelor de Calcul
B-dul Prof. Dimitrie Pompei 6
Bucuresti-Sectorul 2
Tel: 88-20-70, 88-24-40, 88-67-95
Telex: 118

SAUDI ARABIA
Modern Electronic
Establishment (Head Office)
P.O. Box 1228, Baghdadiah Street
Jeddah
Tel: 27 798
Telex: 40035
Cable: ELECTA JEDDAH

Modern Electronic
Establishment (Branch)
P.O. Box 2728
Riyadh
Tel: 62596/66232
Cable: RAOUFCO

Modern Electronic
Establishment (Branch)
P.O. Box 193
Al-Khobar
Tel: 44678-44813
Telex: 67044 OTESTA
Cable: ELECTA AL-KHOBAR

SPAIN
Hewlett-Packard Española, S.A.
Calle Jerez 3
E-**Madrid** 3
Tel: (1) 458 26 00 (10 lines)
Telex: 23515 hpe

Hewlett-Packard Española, S.A.
Milanesado 21-23
E-**Barcelona** 17
Tel: (3) 203 6200 (5 lines)
Telex: 52603 hpbe e

Hewlett-Packard Española, S.A.
Av Ramón y Cajal, 1
Edificio Sevilla, planta 9°
Seville 5
Tel: 64 44 54/58

Hewlett-Packard Española S.A.
Edificio Albia II 7° B
E-**Bilbao** 1
Tel: 23 83 06/23 82 06

Hewlett-Packard Española S.A.
C/Ramon Gordillo 1
(Entlo.)
E-**Valencia**-10
Tel: 96-361.13.54/361.13.58

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 3, Fack
S-161 **Bromma** 20
Tel: (08) 730 05 50
Telex: 10721
Cable: MEASUREMENTS
Stockholm

Hewlett-Packard Sverige AB
Frötallsgatan 30
S-421 32 **Västra Frölunda**
Tel: (031) 49 09 50
Telex: 10721 via Bromma office

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
P.O. Box 307
CH-8952 **Schlieren-Zurich**
Tel: (01) 7305240
Telex: 53933 hpag ch
Cable: HPAG CH

Hewlett-Packard (Schweiz) AG
Château Bloc 19
CH-1219 **Le Lignon-Geneva**
Tel: (022) 96 03 22
Telex: 27333 hpag ch
Cable: HEWPACKAG Geneva

SYRIA
General Electronic Inc.
Nuri Basha-Ahnaf Ebn Kays Street
P.O. Box 5781
Damascus
Tel: 33 24 87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS

Medical/Personal Calculator only
Sawah & Co.
Place Azmé
B.P. 2308
Damascus
Tel: 16 367-19 697-14 268
Telex: 11304 SATACO SY
Cable: SAWAH, DAMASCUS

Suleiman Hilal El Mlawi
P.O. Box 2528
Mamoun Bitar Street, 56-58
DamascusTel: 11 46 63
Telex: 11270
Cable: HILAL DAMASCUS

TUNISIA
Tunisie Electronique
31 Avenue de la Liberte
Tunis
Tel: 280 144

Corema
1 ter. Av. de Carthage
Tunis
Tel: 253 821
Telex: 12319 CABAM TN

TURKEY
TEKNIM Company Ltd.
Riza Sah Pehlevi
Caddesi No. 7
Kavaklidere, **Ankara**
Tel: 275800
Telex: 42155 TKNM TR

Medical only
E.M.A.
Muhendislik Kollektif Sirketi
Mediha Eldem Sokak 41/6
Yüksel Caddesi
Ankara
Tel: 17 56 22
Cable: EMATRADE/Ankara

Analytical only
Yilmaz Ozyurek
Milli Mudafaa Cad 16/6
Kizilay
Ankara
Tel: 25 03 09 - 17 80 26
Telex: 42576 OZEK TR
Cable: OZYUREK ANKARA

UNITED ARAB EMIRATES
Emitac Ltd.
P.O. Box 1641
Sharjah
Tel: 24121-3
Telex: 8136 EMITAC SH
Cable: EMITAC SHARJAH

UNITED KINGDOM
Hewlett-Packard Ltd.
King Street Lane
GB-**Winnersh,** Wokingham
Berks. RG11 5AR
Tel: (0734) 78 47 74
Cable: Hewpie London
Telex:847178/9

Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
Altrincham
Cheshire WA14 1NU
Tel: (061) 928 6422
Telex: 668068

Hewlett-Packard Ltd.
Lygon Court
Hereward Rise
Dudley Road
Halesowen,
West Midlands B62 8SD
Tel: (021) 550 9911
Telex: 339105

Hewlett-Packard Ltd.
Wedge House
799, London Road
GB-**Thornton Heath**
Surrey CR4 6XL
Tel: (01) 684 0103/8
Telex: 946825

Hewlett-Packard Ltd
10, Wesley St.
Castleford
Yorks WF10 1AE
Tel: (0977) 550016
Telex: 557355

Hewlett-Packard Ltd
1, Wallace Way
GB-**Hitchin**
Hertfordshire, SG4 OSE
Tel: (0462) 31111
Telex: 82.59.81

Hewlet-Packard Ltd
2C, Avonbeg Industrial Estate
Long Mile Road
Dublin 12
Tel: Dublin 514322/514224
Telex: 30439

USSR
Hewlett-Packard
Representative Office USSR
Pokrovsky Boulevard 4/17-kw 12
Moscow 101000
Tel: 207.59.24
Telex: 7825 hewpak su

YUGOSLAVIA
Iskra-Standard/Hewlett-Packard
Miklosiceva 38/VII
61000 **Ljubljana**
Tel: 31 58 79/32 16 74
Telex: 31583

SOCIALIST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard Ges.m.b.H
Handelskai 52
P.O. Box 7
A-1205 **Vienna**, Austria
Tel: (0222) 35 16 21 to 27
Cable: HEWPAK Vienna

MEDITERRANEAN AND MIDDLE EAST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard S.A.
Mediterranean and Middle
East Operations
35, Kolokotroni Street
Platia Kefallariou
GR-Kifissia-**Athens**, Greece
Tel: 8080337/359/429
Cable: HEWPACKSA Athens

FOR OTHER AREAS NOT LISTED CONTACT
Hewlett-Packard S.A.
7, rue du Bois-du-Lan
P.O. Box
CH-1217 Meyrin 2 - **Geneva**
Switzerland
Tel: (022) 82 70 00
Cable: HEWPACKSA Geneva

UNITED STATES

ALABAMA
P.O. Box 4207
8290 Whitesburg Dr.
Huntsville 35802
Tel: (205) 881-4591

8933 E. Roebuck Blvd.
Birmingham 35206
Tel: (205) 836-2203/2

ARIZONA
2336 E. Magnolia St.
Phoenix 85034
Tel: (602) 244-1361

2424 East Aragon Rd.
Tucson 85706
Tel: (602) 889-4661

***ARKANSAS**
Medical Service Only
P.O. Box 5646
Brady Station
Little Rock 72215
Tel: (501) 376-1844

CALIFORNIA
1430 East Orangethorpe Ave.
Fullerton 92631
Tel: (714) 870-1000

3939 Lankershim Boulevard
North Hollywood 91604
Tel: (213) 877-1282
TWX: 910-499-2671

5400 West Rosecrans Blvd.
P.O. Box 92105
World Way Postal Center
Los Angeles 90009
Tel: (213) 970-7500
TWX: 910-325-6608

***Los Angeles**
Tel:·(213) 776-7500

3003 Scott Boulevard
Santa Clara 95050
Tel: (408) 249-7000
TWX: 910-338-0518

***Ridgecrest**
Tel: (714) 446-6165

646 W. North Market Blvd
Sacramento 95834
Tel: (916) 929-7222

9606 Aero Drive
P.O. Box 23333
San Diego 92123
Tel: (714) 279-3200

***Tarzana**
Tel: (213) 705-3344

COLORADO
5600 DTC Parkway
Englewood 80110
Tel: (303) 771-3455

CONNECTICUT
12 Lunar Drive
New Haven 06525
Tel: (203) 389-6551
TWX: 710-465-2029

FLORIDA
P.O. Box 24210
2727 N.W. 62nd Street
Ft. Lauderdale 33309
Tel: (305) 973-2600

4428 Emerson Street
Unit 103
Jacksonville 32207
Tel: (904) 725-6333

P.O. Box 13910
6177 Lake Ellenor Dr.
Orlando 32809
Tel: (305) 859-2900

P.O. Box 12826
Suite 5, Bldg. 1
Office Park North
Pensacola 32575
Tel: (904) 476-8422

GEORGIA
P.O. Box 105005
450 Interstate North Parkway
Atlanta 30348
Tel: (404) 955-1500
TWX:810-766-4890

Medical Service Only
***Augusta** 30903
Tel: (404) 736-0592

P.O. Box 2103
1172 N. Davis Drive
Warner Robins 31098
Tel: (912) 922-0449

HAWAII
2875 So. King Street
Honolulu 96826
Tel: (808) 955-4455

ILLINOIS
5201 Tollview Dr.
Rolling Meadows 60008
Tel: (312) 255-9800
TWX: 910-687-2260

INDIANA
7301 North Shadeland Ave.
Indianapolis46250
Tel: (317)842-1000
TWX: 810-260-1797

IOWA
2415 Heinz Road
Iowa City 52240
Tel: (319) 338-9466

KENTUCKY
Medical Only
3901 Atkinson Dr.
Suite 407 Atkinson Square
Louisville 40218
Tel: (502) 456-1573

LOUISIANA
P.O. Box 1449
3229-39 Williams Boulevard
Kenner 70063
Tel: (504) 443-6201

MARYLAND
6707 Whitestone Road
Baltimore 21207
Tel: (301) 944-5400
TWX: 710-862-9157

2 Choke Cherry Road
Rockville 20850
Tel: (301) 948-6370
TWX: 710-828-9684

MASSACHUSETTS
32 Hartwell Ave.
Lexington 02173
Tel: (617) 861-8960
TWX: 710-326-6904

MICHIGAN
23855 Research Drive
Farmington Hills 48024
Tel: (313) 476-6400

724 West Centre Ave.
Kalamazoo 49002
Tel: (606) 323-8362

MINNESOTA
2400 N. Prior Ave.
St. Paul 55113
Tel: (612) 636-0700

MISSISSIPPI
322 N. Mart Plaza
Jackson 39206
Tel: (601) 982-9363

MISSOURI
11131 Colorado Ave.
Kansas City 64137
Tel: (816) 763-8000
TWX: 910-771-2087

1024 Executive Parkway
St. Louis 63141
Tel: (314) 878-0200

NEBRASKA
Medical Only
7I7I Mercy Road
Suite II0
Omaha 68106
Tel: (402) 392-0948

NEVADA
***Las Vegas**
Tel: (702) 736-6610

NEW JERSEY
W. 120 Century Rd.
Paramus 07652
Tel: (201) 265-5000
TWX: 710-990-4951

Crystal Brook Professional
Building, Route 35
Eatontown 07724
Tel:(201) 542-1384

NEW MEXICO
P.O. Box 11634
Station E
11300 Lomas Blvd., N.E.
Albuquerque 87123
Tel: (505) 292-1330
TWX: 910-989-1185

156 Wyatt Drive
Las Cruces 88001
Tel: (505) 526-2484
TWX: 910-9983-0550

NEW YORK
6 Automation Lane
Computer Park
Albany 12205
Tel: (518) 458-1550
TWX: 710-444-4961

650 Perinton Hill Office Park
Fairport 14450
Tel: (716) 223-9950
TWX: 510-253-0092

5858 East Molloy Road
Syracuse 13211
Tel: (315) 455-2486

1 Crossways Park West
Woodbury 11797
Tel: (516) 921-0300
TWX: 510-221-2183

NORTH CAROLINA
5605 Roanne Way
Greensboro 27405
Tel: (919) 852-1800

OHIO
Medical/Computer Only
Bldg. 300
1313 E. Kemper Rd.
Cincinnati 45426
Tel: (513) 671-7400

16500 Sprague Road
Cleveland 44130
Tel: (216) 243-7300
TWX: 810-423-9430

330 Progress Rd.
Dayton 45449
Tel: (513) 859-8202

1041 Kingsmill Parkway
Columbus 43229
Tel: (614) 436-1041

OKLAHOMA
P.O. Box 32008
6301 N. Meridan Avenue
Oklahoma City 73112
Tel: (405) 721-0200

4110 S. 100th E. Avenue
Grant Bldg.
Tulsa 74145

OREGON
17890 SW Lower Boones
Ferry Road
Tualatin 97062
Tel: (503) 620-3350

PENNSYLVANIA
111 Zeta Drive
Pittsburgh 15238
Tel: (412) 782-0400

1021 8th Avenue
King of Prussia Industrial Park
King of Prussia 19406
Tel: (215) 265-7000
TWX: 510-660-2670

PUERTO RICO
Hewlett-Packard Inter-Americas
Puerto Rico Branch Office
Calle 272,
Edif. 203 Urg. Country Club
Carolina 00924
Tel: (809) 762-7255
Telex: 345 0514

SOUTH CAROLINA
P. O. Box 6442
6941-0 N. Trenholm Road
Columbia 29260
Tel: (803) 782-6493

TENNESSEE
8914 Kingston Pike
Knoxville 37922
Tel: (615) 523-0522

3027 Vanguard Dr.
Director's Plaza
Memphis 38131
Tel: (901) 346-8370

***Nashville**
Medical Service only
Tel: (615) 244-5448

TEXAS
4171 North Mesa
Suite C110
El Paso 79902
Tel: (915) 533-3555

P.O. Box 1270
201 E. Arapaho Rd.
Richardson 75080
Tel: (214) 231-6101

P.O. Box 42816
10535 Harwin Dr.
Houston 77036
Tel: (713) 776-6400

***Lubbock**
Medical Service only
Tel: (806) 799-4472

205 Billy Mitchell Road
San Antonio 78226
Tel: (512) 434-8241

UTAH
2160 South 3270 West Street
Salt Lake City 84119
Tel: (801) 972-4711

VIRGINA
P.O. Box 12778
No. 7 Koger Exec. Center
Suite 212
Norfolk 23502
Tel: (804) 461-4025/6

P.O. Box 9669
2914 Hungary Springs Road
Richmond 23228
Tel: (804) 285-3431

WASHINGTON
Bellefield Office Pk.
1203-114th Ave. S.E.
Bellevue 98004
Tel: (206) 454-3971
TWX: 910-443-2446

***WEST VIRGINIA**
Medical/Analytical Only
Charleston
Tel: (304) 345-1640

WISCONSIN
9004 West Lincoln Ave.
West Allis 53227
Tel: (414) 541-0550

FOR U.S. AREAS NOT LISTED:
Contact the regional office
nearest you: Atlanta, Georgia...
North Hollywood, California...
Rockville, Maryland...Rolling Meadows,
Illinois.Their complete
addresses are listed above.

***Service Only** 8/78

Part No. 32215-90003
Printed in U.S.A. 4/78
3IMAGEA.320.32215-90003

HEWLETT hp PACKARD

Sales and service from 172 offices in 65 countries.
5303 Stevens Creek Blvd., Santa Clara, California 95050