HEWLETT hp PACKARD

HP

32215A

IMAGE/3000
Data Base Management System
reference and application manual

HP 3000 COMPUTER SYSTEMS

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP 32215A

IMAGE/3000
Data Base Management System

reference and application manual

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

MANUAL PART NO. 32215-90001
MICROFICHE PART NO. 32215-90002

Printed: AUG 1974
Printed in U.S.A.

LIST OF EFFECTIVE PAGES

Pages	Effective Date
Title	Aug 1974
ii to vi	Aug 1974
1-1 to 1-9	Aug 1974
2-1 to 2-4	Aug 1974
3-1 to 3-2	Aug 1974
4-1 to 4-11	Aug 1974
5-1 to 5-12	Aug 1974
6-1 to 6-5	Aug 1974
A-1 to A-17	Aug 1974
B-1 to B-5	Aug 1974
C-1 to C-7	Aug 1974
D-1 to D-2	Aug 1974
E-1 to E-2	Aug 1974
F-1	Aug 1974

PREFACE

This manual describes the IMAGE/3000 Data Base Management System for HP 3000 computers. It is the reference document for both the data base manager who designs and maintains a data base and the applications programmer who writes programs to access a data base.

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

Section I of this document describes the components of the IMAGE/3000 product, as well as the logical organization of data bases created by IMAGE/3000. Section II describes internal techniques of IMAGE/3000 operation, such as the manner in which file space is allocated and the methods used to link related data in the data base. Section III discusses privacy and security and the ways which the data base designer can control access to the data base. The data base description language and the IMAGE/3000 program which processes that language (Schema Processor) is described in Section IV, while Section V outlines the use of the IMAGE/3000 library procedures for data base access. Section VI describes the IMAGE/3000 utility programs that create backup copies of the data base and perform other utility functions for the data base designer. A sample data base application is shown in Appendix A, to give the data base designer a comprehensive example of a complete IMAGE/3000 data base. Appendices B, C, and D list the error conditions and related error messages which may occur during Schema Processor, library procedures, or utility program operation respectively. Appendix E discusses the rules for restructuring already existing data bases, while Appendix F details some of the backup and recovery methods that can be used to protect the data base from hardware or software failures.

CONTENTS

CONTENTS (continued)

ILLUSTRATIONS

TABLES

INTRODUCTION

SECTION I

IMAGE/3000 is designed to create, access, and maintain logically-related files which make up a data base. These files contain both data and structure information. Pointers are maintained by IMAGE/3000 to relate data within the files and to index data across files. The indexing property of IMAGE/3000 data bases provides the ability to rapidly access related data in the data base.

The data base designer specifies the data base structure using the IMAGE/3000 data base description language. This language defines all aspects of the data base structure including:

- Data items — the smallest accessible units of data
- Data entries — groups of data items
- Data sets — collections of like-defined data entries. Data sets reside in the disc files of the data base.
- Security provisions — to control access to subsets of information in the data base.

```
BEGIN DATA BASE COMPNY;

 LEVELS;
  3 TRICIA;
  9 NARTHEX;

 ITEMS;
  BADGE#,   I2;
  DATE,     X8;
  DEPT,     X4;
  DESCRIP,  X16(9,10);
  HOURS,    R2;
  PROJECT,  X4;
  SALARY,   P8 (15,15);
  SEX,      X2;

 SETS;

  NAME;     EMPLOYEE,MANUAL(3,14);
  ENTRY;    BADGE#(1),
            DEPT,
            SALARY,
            SEX;
  CAPACITY; 500;

  NAME;     PROJECT-MASTER,MANUAL(3,14);
  ENTRY;    PROJECT(1),
            DESCRIP;
  CAPACITY; 75;

  NAME;     LABOR,DETAIL(3,9);
  ENTRY;    BADGE#(EMPLOYEE(PROJECT)),
            PROJECT(PROJECT-MASTER),
            HOURS,
            DATE;
  CAPACITY; 10000;

END.
```

Figure 1-1. Sample of Data Base Description Language

Figure 1-1 shows a sample of the data base description language. The specifics of the language are outlined in Section IV.

Access to the data base occurs through the IMAGE/3000 library procedures. These procedures are invoked through CALL statements imbedded in host-language user application programs. The programs can be written in HP 3000 FORTRAN, COBOL, or SPL. The library procedures locate data, maintain pointer information, manage the allocated file space, and return status information to the user. The tasks performed by the library procedures relieve the programmer of the "bookkeeping" normally associated with file management and allow him to concentrate on using the processing power of the host language. Figure 1-2 shows a sample CALL statement as it might appear in a FORTRAN program. The IMAGE/3000 library procedures and their calls are discussed in Section V.

Maintenance of the data base is provided by utility programs. These programs help maintain data base integrity by providing magnetic tape backup copies of the data base as well as performing other utility functions. The utility programs are completely described in Section VI.


```
CALL DBGET
(BASE, DSET, MODE, STATUS, LIST, BUFFER, ARGUM)
```

Figure 1-2. Sample CALL Statement

1-1. COMPONENTS

IMAGE/3000 consists of three components which run under the control of the MPE/3000 operating system in either Job (batch) or Session (interactive) mode.

- The Schema Processor processes a description of the data base and stores the data base requirements on a disc file.

- IMAGE/3000 library procedures are invoked by user-written application programs for the purpose of modifying and retrieving data base content.
- A set of utility programs provide maintenance functions such as copying a data base to tape.

1-2. USING IMAGE/3000

Using the IMAGE/3000 product involves five major steps:

a. Creating an external data base description
b. Building the internal data base description
c. Building the data base files
d. Accessing the data base
e. Maintaining the data base

1-3. CREATING AN EXTERNAL DATA BASE DESCRIPTION

The designer describes the data base using the IMAGE/3000 data base description language. This description is called a **schema**. The schema describes all aspects of the data base design including the smallest elements of accessible data, the number of files, and the relationship between files. In addition, the schema describes the security provisions for the data base.

1-4. BUILDING THE INTERNAL DATA BASE DESCRIPTION

To build an internal description of the data base, the designer invokes the Schema Processor. The Schema Processor accepts the schema as input and creates an internal representation of the schema on a disc file known as the **root file.**

1-5. BUILDING THE DATA BASE FILES

The designer invokes a utility program to build empty data files on disc, one file for each defined data set.

1-6. ACCESSING THE DATA BASE

User application programs call IMAGE/3000 library procedures to enter, retrieve, and modify data base content.

1-7. MAINTAINING THE DATA BASE

To maintain the data base, the designer may invoke IMAGE/3000 utility programs to perform the following functions.

a. Copy a data base from disc files to magnetic tape.
b. Reload a magnetic tape data base copy back onto data base disc files.
c. Erase data base content while maintaining the root file and data files of the data base.
d. Purge the entire data base including content, root file, and data files.

The designer can copy either the data base content or the data base content and related structure information to magnetic tape. The first option is useful for restructuring data bases while keeping the content intact. The exact steps for doing so are described in Appendix E.

1-8. IMAGE/3000 DATA ELEMENTS

1-9. DATA ITEMS

The **data item** is the smallest accessible data element. Each data item consists of a value referenced by a data item name. Data item names are character strings, up to 16 characters in length, typically selected to suggest an interpretation of the referenced values. In general, many data item values are referenced by the same data item name. Figure 1-3 shows an example of a data item name with three values.

The maximum length for a data item value is 2047 16-bit computer words.

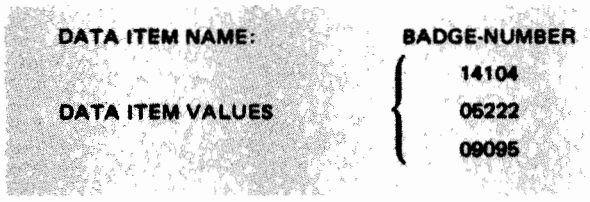

Figure 1-3. Data Items

1-10. DATA ENTRIES

A **data entry** is an ordered set of related data items. The order of data item values in an entry is determined by the ordered list of data item names defining the data entry. The length of the data entry is the combined length of member data items.

Data entries may be defined with at most 127 data item names, none of which is repeated. Figure 1-4 shows a sample data entry.

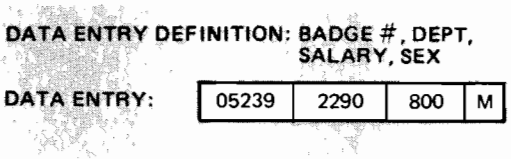

Figure 1-4. Sample Data Entry

1-11. DATA SETS

A **data set** is a collection of data entries sharing a common definition. Each data set is referenced by a unique **data set name** consisting of a character string up to 16 characters in length. Figure 1-5 shows an example of three data entries in a data set named EMPLOYEE:

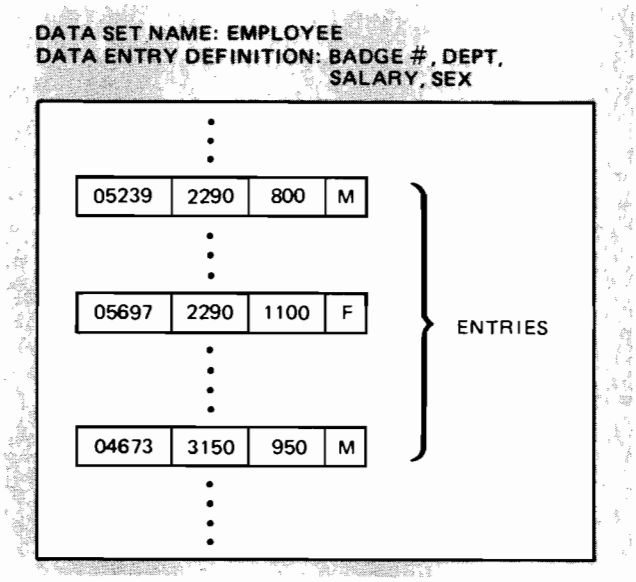

Figure 1-5. Sample Data Set

Data sets are stored in disc files consisting of storage locations (called records). The number of records in each data set depends upon the capacity defined for the data set in the data base schema, but the maximum number of records allowed depends upon the record size as well as the available disc space and file system constraints. Each record is identified by a record number which can be used to access the entry within it. The numbers are integers from 1 to **n**, where n is the defined capacity of the data set.

Record sizes vary between data files but are constant within each file. The record is large enough to contain a data entry and any associated IMAGE/3000 pointer information. The amount of pointer information associated with a data entry depends upon the data set's defined type and relation to other data sets. Pointer information is described later in this section and in Section II.

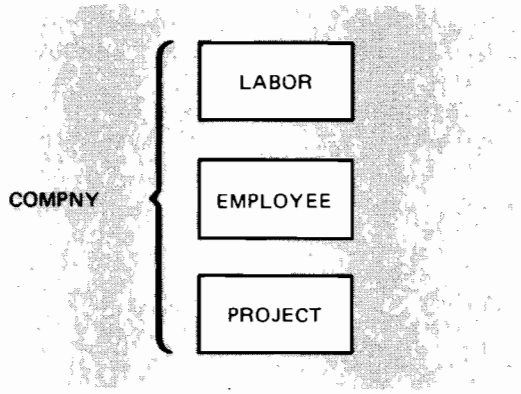

Figure 1-6. Sample Data Base

1-12. DATA BASES

A **data base** is a named collection of related data sets. The data base is defined in terms of data items and data sets. A data base is referenced by a **data base name** consisting of a character string up to six characters in length. The data base may be defined with a maximum of 255 data item names and 99 data set names. An example of a data base named COMPNY consisting of three data sets named LABOR, EMPLOYEE, and PROJECT is shown in Figure 1-6.

1-13. TYPES OF DATA SETS

1-14. MASTER DATA SETS

Master data sets are characterized in two ways:

- The storage location assigned to each master entry is determined by the value of a specific data item within the entry. The data item so used is called a **search item** or **key**. The exact method of determining the storage location is discussed in Section II.
- All master entries contain unique search item values. The search item value therefore uniquely identifies the data entry.

The above two characteristics permit users to retrieve master entries using the calculated access method (described later in this section) based upon the value of the search item. Master data sets are employed whenever a single data entry corresponds to a uniquely identifiable event and rapid retrieval based upon the identifier is desired. Figure 1-7, for example, shows a master data set used to store employee information. Each master entry contains information about a different employee. The employee's badge number is the designated search item.

MASTER DATA SET NAME: EMPLOYEE
DATA ENTRY DEFINITION: BADGE #, DEPT, SALARY, SEX
SEARCH ITEM NAME: BADGE #

1	04673	2256	0900	F
2	05999	7400	1000	M
3				
4	21532	1352	2700	F
5				
6	04379	6239	1000	F
7				
8				
9				
10	35286	2256	1500	M
11	61704	2256	1150	M
12				
13				

Figure 1-7. Sample Master Data Set

DETAIL DATA SET NAME: LABOR
DATA ENTRY DEFINITION: BADGE #, PROJECT, HOURS, DATA
SEARCH ITEM NAMES: BADGE #, PROJECT

ENTRY NUMBER				
1	04673	233	4.5	02/27/74
2	04673	254	3.5	02/27/74
3	21532	233	8	02/27/74
4	04673	254	8	02/28/74
5	35286	207	4	02/27/74
6	21532	207	8	03/04/74
7				
8	21532	299	2.5	03/01/74
9	04673	207	6	03/05/74
10				
11				
12				

Figure 1-8. Sample Detail Data Set

The entry containing badge number 21532 may be retrieved with a single call to the appropriate library procedure. Although there are unused storage locations in the data set, IMAGE/3000 disallows any attempt to add another data entry with a value of 21532 for BADGE#. The search item value of each entry must remain unique. Note, however, that the values of other data items in the master data set are not necessarily unique. This is allowable because the data items are not search items and are not used for calculated access to the data set.

1-15. DETAIL DATA SETS

A **detail data set** is a data set in which the storage location assigned to a particular 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 master data sets, 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 characteristics of detail data sets enable their use in recording information about related events. For example, Figure 1-8 shows a detail data set used to hold information about the amount of time employees spend on various projects. Each entry contains the number of hours worked on a given day by a specific employee on a particular project. BADGE# and PROJECT are designated as search items, and their values are not always unique, as is shown in the figure. Information about the same employee or the same project may be contained in more than one entry.

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. IMAGE/3000 stores pointer information with each data entry such that entries with the same search item value are linked together, although the entries may not be physically adjacent to each other. (Pointer information is described in greater detail in Section II.)

A search item defined for a detail data set forms a **path**. A path exists for each search item defined. For a defined search item, entries which share a common search item value are linked together through IMAGE/3000 pointer information to form a **chain**. A path contains a chain for each unique search item value. An entry belongs to exactly one chain for each path defined, and separate chain linkage information is maintained in an entry for each chain.

In the detail data set illustrated in Figure 1-8, data items BADGE# and PROJECT are designated as search items, defining two paths. The BADGE# path contains three chains, one for each unique badge number. Entries 3, 6, and 8 are members of the BADGE# = 21532 chain. The PROJECT path contains four chains. Entries 1 and 3 are members of the PROJECT = 233 chain.

Chained access to a detail data set consists of successive retrieval of the member data entries of a specified chain. Chained access is possible only on detail data sets for which search items are defined. IMAGE/3000 maintains and uses pointer information to directly locate members of a chain. This is achieved by first locating the beginning of the chain and then successively reading the individual entries of the chain. In Figure 1-8, for example, chained access permits rapid retrieval of all those pieces of work

done by the employee whose badge number is 21532, or all of the work done on project 207. The retrieved entries can then be examined by the user's program to discover (for example) the total number of hours worked by employee 21532, or the badge numbers of all employees who worked on project 207. By contrast, a program using a non-search item such as DATE to locate entries containing specific DATE values must serially search the data set. The selection of data items as search items depends upon expected usage, frequency of retrieval, number of distinct values, and other criteria.

For any path, it is possible to designate some data item other than the search item as a **sort item.** In this case, each of the chains of this path are maintained in 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.

When a new entry is added to a detail data set, it is physically placed in an available record location. The new entry is also logically inserted into each sorted chain in such a manner that subsequent retrieval of the members of the chain will be in sort item order. If a sort item is not designated for a path, new entries are appended to the end of its chains.

In Figure 1-8, for example, PROJECT is designated as the sort item for the BADGE# path. Therefore, the BADGE# = 21532 chain is ordered: entry 6, entry 3, entry 8.

1-16. DATA SET INTERACTIONS

1-17. INDEXING PROPERTIES OF MASTER DATA SETS

Within a data base description, each path of a detail data set is specified by a search item name (belonging to the detail data set), a master data set name, and optionally a sort item (if sorted chains are desired). The master data set name defines a relationship between the named master data set and the detail data set.

IMAGE/3000 maintains and uses pointer information associated with each master data set entry. This information consists of pointers to the first and last entries of a specific chain in each related detail data set. One set of pointers is maintained for each relationship defined between the master data set and related detail data sets. Pointers which locate the first and last members of a chain are maintained within the related master data set in the master data entry whose search item value matches the chain's search item value.

To perform chained access on detail entries for a specific search item value, IMAGE/3000 first performs calculated access to locate the appropriately related master data entry containing the desired search item value. Then IMAGE/3000 uses the pointers associated with the master entry to locate the first and last detail entries in the chain. These starting record addresses together with the bi-directional pointers associated with each entry of the chain enable subsequent forward or backward access to all the members of the chain.

A master data set related to a detail data set serves as an index to the chains of a path. The indexing function of a master data set and its search item values is similar to a library catalog. Multiple indexing of a detail data set by one or more master data sets is analogous to multiple cataloging in a library. For example, Figure 1-9 depicts a detail data set named LABOR for which search items named BADGE# and PROJECT are defined. The master data sets EMPLOYEE and PROJECT-MASTER contain a search item (key) named BADGE# and PROJECT, respectively. EMPLOYEE is the master data set associated with the BADGE# path in LABOR, and PROJECT-MASTER is associated with the PROJECT path. Note that chains of the BADGE# path are sorted according to the value of PROJECT (i.e., PROJECT is defined as a sort item for the BADGE# path).

In Figure 1-9, the master data set named EMPLOYEE and the master data set named PROJECT-MASTER contain data items not defined as search items (and are therefore not used for indexing). In each master data set entry, these data items contain information common to the chain indexed by the entry. In the master data set named EMPLOYEE, the entry for the employee with badge number 04673 contains the employee's department number, salary, and sex.

To the left of the data portion of each data entry in the two master data sets is a representation of the chain information stored with the entry. This consists of the record addresses of the first and last detail entries in the associated chain, together with the total number of entries in the chain. For example, the LABOR data set contains four data entries belonging to the BADGE# = 04673 chain. The first member of the chain is entry 9 and the last is entry 4. Entry 9 is also the last entry of the PROJECT = 207 chain, as is shown in the PROJECT = 207 entry in PROJECT-MASTER. This chain consists of three member entries, the first of which is number 5.

Some of the entries of EMPLOYEE have key values which do not occur as search item values (for BADGE#) in the LABOR data set. The chain lengths for these values are zero. The reverse situation, a detail search item value with no corresponding key item value in the related master, is not permitted. Figure 1-9 does not show the chain pointers within the detail data set. These are discussed in Section II.

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, independent chain information like that illustrated in Figure 1-9 is kept with each master

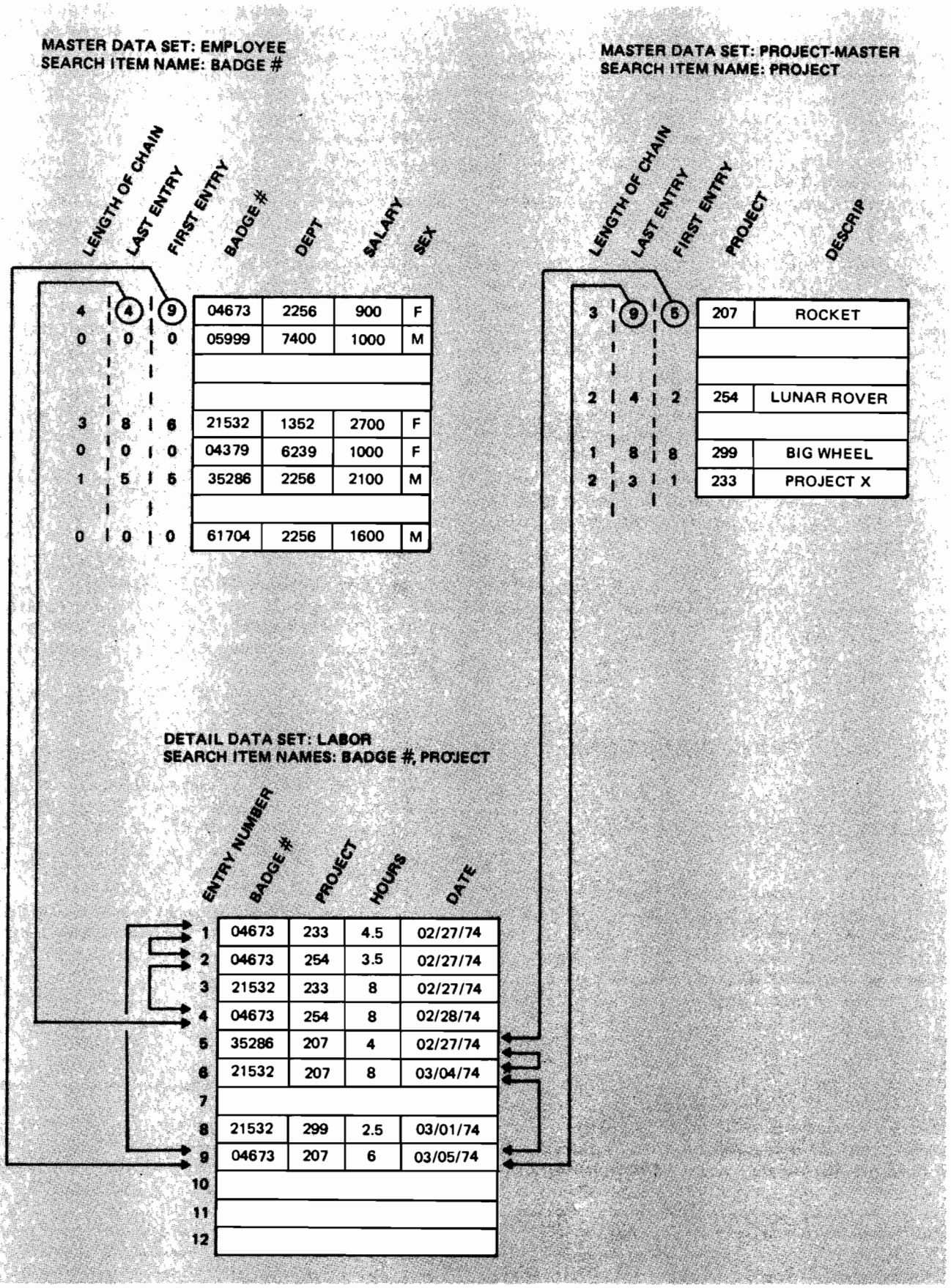

Figure 1-9. Master/Detail Data Set Relationships

entry. Each detail data set may be singly or multiply indexed by one ore more master data sets. No detail data set may be involved in more than 16 such relationships and each such relationship must be relative to a different search item.

1-18. MANUAL AND AUTOMATIC MASTER DATA SETS

Within a data base description, each master data set may be defined as **manual** or **automatic**. These two types of masters have the following characteristics.

Manual Masters:

- May be stand-alone (i.e., not related to any detail data set)
- May contain data items in addition to the search item

Automatic Masters:

- Must be related to one or more detail data sets (i.e., cannot stand alone).
- Must contain only one data item, namely the search item.

Non stand-alone manual masters differ optionally from automatic masters in the following ways:

Manual Masters:

- All entries are explicitly added or deleted by the user.
- An entry may be added to related details only if a master entry with matching key exists.
- The key values of existing master entries serve as a table of legitimate search item values for all related detail data sets.

Automatic Masters:

- All entries are implicitly added by IMAGE/3000 when needed and deleted when no longer needed.
- Addition of an entry to a related detail with a search item value different from all current key values of the master causes an automatic addition of a master entry with a matching key value.
- Deletions of detail entries triggers an automatic deletion of the matching master if it is determined that all related data chains are empty.

In the example of Figure 1-9, any attempt to add an entry to the LABOR data set whose values of BADGE# and PROJECT did not match an existing BADGE# of EMPLOYEE and an existing PROJECT of PROJECT-MASTER is disallowed by IMAGE/3000. This illustrates the editing characteristic of manual masters. It is recommended that data base designers take advantage of this feature to prevent entry of erroneous data by using manual masters in preference to automatic masters. In some instances, automatic masters can be extremely useful and time saving, especially when the search item values of related detail entries 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 input editing of manual masters and the time savings offered by automatic masters.

For example, Figure 1-10 shows a detail data set LABOR related to the automatic data set DATE-MASTER through the detail search item called DATE. (Note that this is similar to the situation in Figure 1-9, except that an additional data set has been added.)

Entries are deleted or added automatically to DATE-MASTER as needed. For example, if entry 8, which is the only member of the DATE = 03/01/74 chain, is deleted from LABOR, the DATE = 03/01/74 entry is automatically deleted from DATE-MASTER. Similarly, if a new entry is added to LABOR with a DATE value of 27/43/99, an entry with a like key value is added automatically to DATE-MASTER.

1-19. DATA ELEMENT ACCESS

Access to data within a data base occurs through calls to IMAGE/3000 library procedures. Each of the major access functions is carried out by a separate procedure. Some of the functions accomplished through the use of the procedures are:

- Adding a new entry to a data set
- Deleting an entry from a data set
- Reading some or all of the data items of an entry
- Changing the values of data items of a data entry

When one of the procedures is called, the following information must be specified as parameters passed to the procedure by the user program:

- The data base
- The data set
- The data items
- The entry of interest within the specified data set
- The address of a buffer in the user's data area

The data items specified need not appear in the data item list in the order that the data item values appear in the data entry. IMAGE/3000 acts upon only those data items specified regardless of the order they are specified. The user need not be aware of the actual physical ordering of data items within the entry, nor even the physical location of the entry itself.

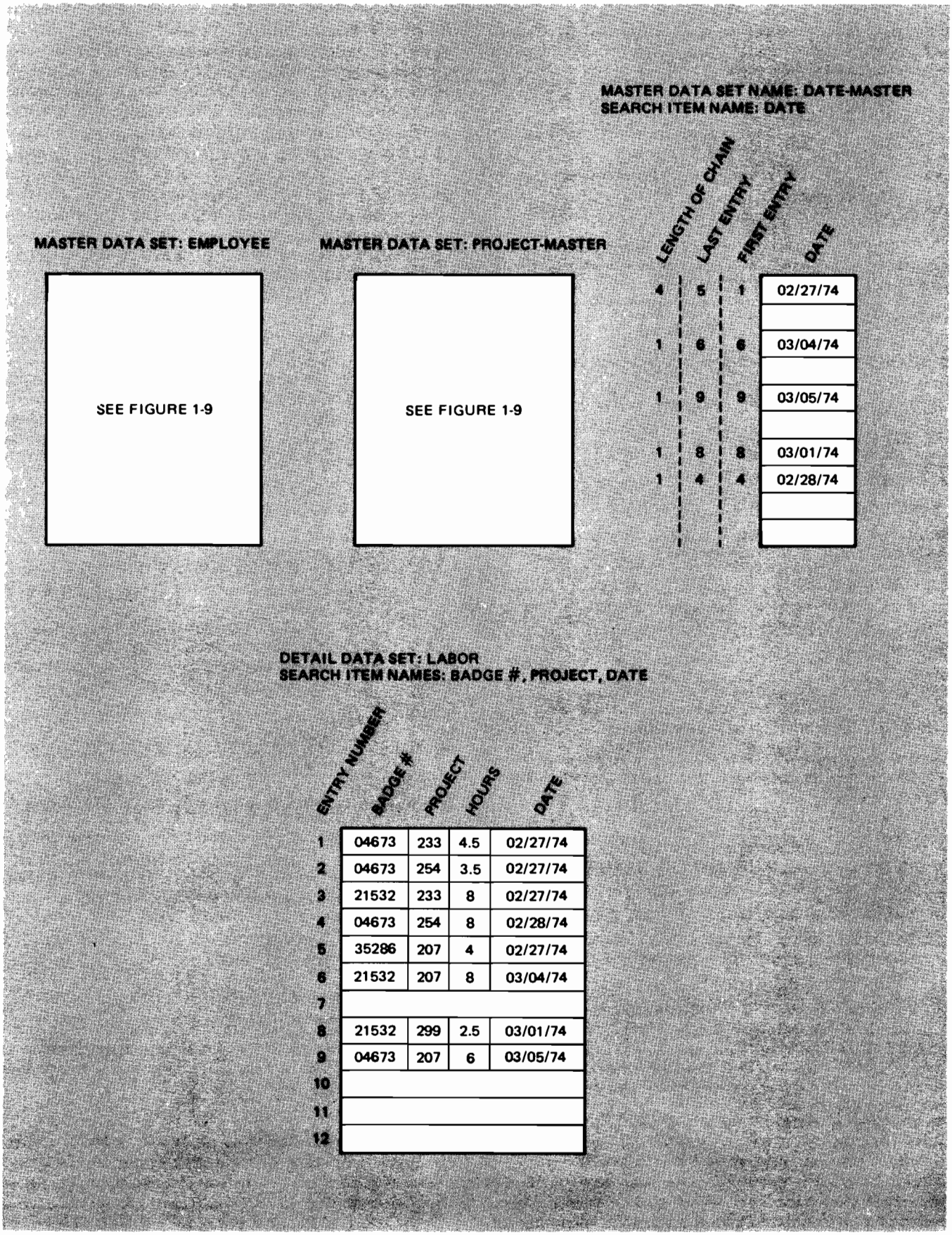

Figure 1-10. Automatic Master Data Set

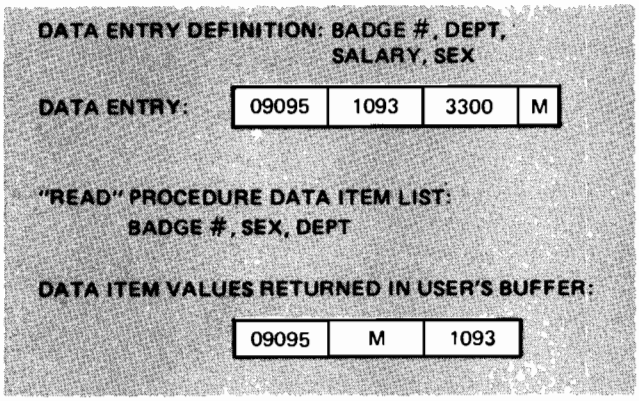

Figure 1-11. Values Returned in User Buffer

For example, the "read" procedure retrieves an entry as it is stored in the data base and transfers to the user's buffer area only those data items required. Figure 1-11 shows the values in a user's buffer area resulting from the read of a specific data entry with a specific data item list. Note that the resulting data item values are returned to the buffer area in the order specified in the data item list regardless of the actual physical order of the values in the data entry.

1-20. DATA ENTRY SELECTION

Data sets exist physically as disc files containing data entry storage locations (called records). An entry is identified by the record address (relative record number) at which it is stored. The first record in the data set is record number 1, and the last record 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/3000 maintains internal information indicating which records of a data set contain entries and which do not.

Each time a request for retrieval is made, the library procedure must determine the data entry from which to extract the desired data items. The determination is based upon the type of access requested by the calling program. The choice is conveyed as a parameter to the library procedure.

1-21. SERIAL ACCESS

The most basic method of entry selection is called **serial access**. In this mode of retrieval, IMAGE/3000 starts at the most recently accessed storage location for the data set, called the **current record**, and sequentially examines adjacent records until the closest entry, if any, is located. Data items from this entry are returned to the calling program, and its location becomes the new current record. Retrieving the next greater-numbered entry is referred to as **forward serial access**; searching lower-numbered records for the next entry is called **backward serial access**. For these two modes of access, if no qualifying entry is located, IMAGE/3000 returns an exceptional condition called an **end-of-file** or **beginning-of-file**, respectively.

When a data set is accessed by a program for the first time, there is no current record for it. A forward serial retrieval searches starting at record 1, and examines records with increasing relative positions. Similarly, a backward serial retrieval begins at the highest numbered record, and searches decreasing record locations.

1-22. DIRECTED ACCESS

A second method of entry selection for any data set is called **directed access**. In this mode, the calling program specifies the record address of the data entry from which the requested data items are to be retrieved. If an entry exists at the specified record location, the requested data items are returned to the calling program. If no such entry exists, the program is notified by an exceptional condition return.

1-23. CALCULATED ACCESS

Calculated access is retrieval of master entries based upon the entry's content. It is not necessary for the user to know where the desired entry is located in the data set. In this form, one of the calling parameters is the value of a data item defined as a search item (also called a key). The library procedure locates the entry in the data set whose search item value matches the desired search item value. The exact technique used to perform calculated access is described in Section II.

1-24. CHAINED ACCESS

Chained access is the retrieval of the next entry in the current chain. For a master data set, this means the next entry in the current synonym chain. (Synonym chains are discussed in Section II.) For a detail data set, chained access refer to the current chain for the detail data set. (Refer back to the discussion of detail data sets in this section.)

INTERNAL STRUCTURES AND TECHNIQUES

SECTION II

In addition to the data elements discussed in the preceding section, data bases consist of a number of structure elements. IMAGE/3000 creates and makes use of these, along with various internal techniques, to provide rapid and efficient access to the data content. This section describes those structures and techniques, an understanding of which is necessary for full comprehension of IMAGE/3000.

2-1. STRUCTURE ELEMENTS OF DATA SETS

2-2. POINTERS

IMAGE/3000 uses **pointers** to link one data set record to another. A pointer is a two word entity (doubleword) containing the relative address of a record within a data set.

2-3. DATA CHAINS

Data chain is a set of detail entries which are bi-directionally linked together by pairs of pointers. All entries having a common search item value are placed in the same data 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.

2-4. MEDIA RECORDS

IMAGE/3000 transfers information to and from a storage location on disc in the form of a media record. The media record in each storage location either contains an entry and related pointer information or is null if no data entry is present.

2-5. 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 (as previously described in Section I). The number of pointer pairs corresponds to the number of paths specified for the data set within the schema. Figure 2-1 shows 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 schema and the second set corresponds to the second path.

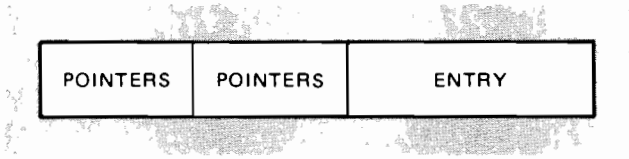

Figure 2-1. Media Record for Detail Entry

2-6. CHAIN HEADS

IMAGE/3000 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 key item value is the same as the search item value defining the chain. Each chain head is five words in length. The first word is an integer count of the number of member entries, possibly zero, in the referenced chain. The remaining four words contain two doubleword pointers. One points to the last chain entry, the other to the first, unless the chain count is zero, in which case both pointers are zero.

2-7. PRIMARY ENTRIES

Selection of record addresses for master entries begins with a calculated address determined by an algorithm applied to the key value of each entry, as 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.**

2-8. SECONDARY ENTRIES

If a new entry with a unique key is assigned the same primary address as an existing primary entry, the two entries are referred to as **synonyms.** IMAGE/3000 assigns the new entry and subsequent synonyms, if any, to a **secondary address** obtained from unused records in the vicinity of the primary entry. All entries residing at secondary addresses are called **secondary entries.**

2-9. SYNONYM CHAINS

A primary entry, together with all of its synonyms, if any, constitutes a **synonym chain**. 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. (The synonym chain head in a primary entry of a master data set is distinct from any detail chain heads which it may contain.)

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 media record for a secondary entry is described in detail below.) 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 synonym chain contains a zero forward pointer.

2-10. MEDIA RECORDS OF MASTER DATA SETS

Media records of master data entries are composed of, in left-to-right order:

- A five-word field serving as a synonym chain head for primary entries or a synonym chain link for secondary entries,
- A 5 x **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 (from zero to 16),
- The entry itself

Figure 2-2 shows the media record for a primary entry of a master data set with one path defined.

Figure 2-2. Media Record for Primary Entry

Figure 2-3 shows a media record for a secondary entry of a master data set with one path defined.

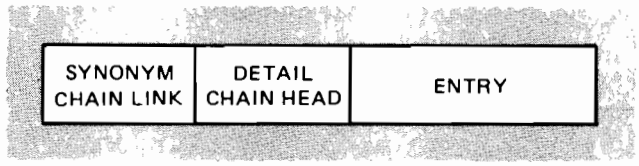

Figure 2-3. Media Record for Secondary Entry

When more than one detail chain head is present, they are physically ordered left-to-right in the order of their specification in the schema.

2-11. BLOCKS

For each data set, media records are physically transferred to and from the disc in groups. Each group of media records involved in a single disc transfer is called a **block**. The number of records in each block is called the **blocking factor**. The blocking factor is chosen by the Schema Processor during creation of the root file with the twin goals of efficient disc space utilization and minimum disc access. The maximum allowable block length is 2048 16-bit computer words. Accompanying each block is a bit map used to inform IMAGE/3000 which of the media records in the block contain entries and which records are empty (not being used). Bit maps are explained in detail below. Figure 2-4 shows a block with a blocking factor of four.

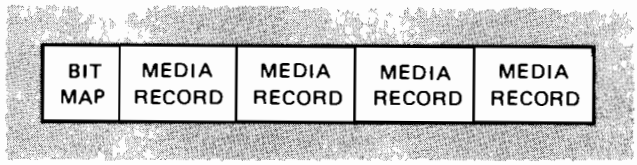

Figure 2-4. Block with Blocking Factor of Four

2-12. BIT MAPS

The first word or words of each block contain a bit map employed by IMAGE/3000 to indicate whether a particular record of the block contains a media entry of the data set. 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 the significant bits. If a bit is zero, the corresponding record is empty. If a bit is one, the record is occupied (contains a data entry and associated structure information.)

2-13. STRUCTURE ELEMENTS OF A DATA BASE

IMAGE/3000 employs privileged MPE/3000 disc files in its implementation of a data base. One file, called a **root file**, is the physical counterpart of the data base definition. Other files called **data files**, contain the data sets of the data base.

Another structure, which exists only during the time a data base is open is called a data base control block (DBCB). It contains data base structure and access information relative to a particular user, and is resident in a privileged extra data segment.

2-14. ROOT FILE

The root file is created by the Schema Processor for the data base creator. It is catalogued, within the creator's log-on group and account, with a local file name identical to the data base name.

The root file is a single extent MPE/3000 disc file. The content of the root file reflects the definition of the data base. The root file serves as a common point of entry to, and source of information about, the data base.

2-15. DATA FILES

There is one data file for each data set of a data base. The file parameters of record size and total number of records of each data file are determined by the contents of the root file. Creation and initialization is accomplished by a utility program on behalf of the data base creator.

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 2-digit suffixes are assigned to the variously named data sets of the data base in keeping with the order of their definition within the schema. The first is assigned 01, the second 02, and so on. For example, a data base SCHOOL with data sets TEACHERS, STUDENTS, and CLASSES would have a root file named SCHOOL and 3 data files named SCHOOL01, SCHOOL02, SCHOOL03.

Each data file is physically constructed from one, two, four, eight or 16 extents of contiguous disc sectors, as needed to meet the capacity requirements of the file, subject to the constraints of the HP 3000 file system. Each data file contains a user label (disc sector) maintained and used by the IMAGE/3000 library procedures. The user label contains structural pointers and counters needed for dynamic storage allocation and deallocation.

2-16. DATA BASE CONTROL BLOCK

A data base control block (DBCB) is a table of data relative information, some static, some dynamic, used by the IMAGE/3000 library procedure in providing and controlling user access to an IMAGE/3000 data base.

The DBCB is created at the time the data base is opened and is destroyed when the data base is closed. During its existence, it resides in a privileged extra data segment, where it serves the same purpose to the data base, and its data sets, as an Access Control Block does to an ordinary MPE/3000 file.

The DBCB consists of a modified version of the root file, four I/O buffers, and one communication area. The modifications relate to the user's access level and access mode. The I/O buffers are shared by all data sets, and each is as large as the largest block of the data base. The communication area is at least as large as the largest entry defined for the data base.

The major dynamic information resides in areas called data set control blocks (DSCB). There is one DSCB for each data set of the data base. Each DSCB contains dynamic status information regarding the user's access to that data set, independent of any accesses to any other data sets. This includes a current record address, a current path, forward and backward pointers for the current chain, and an internal table of the list of data items constituting the last logical record (entry) transferred to or from the user's buffer.

IMAGE/3000 also maintains status information about each I/O buffer, the locked or unlocked state of the data base, and the opened or closed state (both actually and logically) of each data set. All buffer sharing is automatic and data sets are opened and closed automatically as needed. Logical closing of a data set includes resetting the dynamic status information (in the DSCB) to the initial state (as if the data set were not yet accessed).

2-17. INTERNAL TECHNIQUES

2-18. PRIMARY ADDRESS CALCULATION

IMAGE/3000 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 word (see Section IV). The low order (rightmost) 31 bits (or 16 bits if the search item is only one word long) are used to form a 32-bit doubleword value. This doubleword value is then decremented by one, reduced modulo the value representing the data set capacity and incremented by one to form a primary address.

The second method of primary address calculation applies to master data sets with search items of type byte or nibble (see Section IV). In this case, the entire search item value (regardless of length) is used to obtain a positive doubleword (32-bit) value. This doubleword value is then reduced modulo the value representing 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. 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.

2-19. 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 offending 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, along with the attendant linkage and/or chain head maintenance, occurs 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, including its synonym chain head, each synonym chain must have a primary entry. To maintain the integrity of a synonym chain, IMAGE/3000 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 so moved is now empty.

2-20. 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 all of the bit maps, one in each block, of the data set.

Adding a new entry results in the free space counter being decremented and the bit corresponding to the newly assigned record address being changed from a 0 to a 1. The assigned record address is the primary address if the corresponding bit of the bit map is 0. Otherwise, a secondary address, found by a cyclical search of the bit maps for a 0, is assigned.

2-21. 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/3000 assigns to it the record address referenced by the delete chain pointer unless it 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.

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, if any, is always allocated in preference to the unused space represented by the record addresses beyond the end-of-file pointer.

Adding and deleting data entries also involve data chain maintenance and the turning on or turning off, respectively, of 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.

2-22. PRIMARY PATHS

One of the paths, of each detail data set having paths, may be designated by the user as the **primary path.**

The primary path is employed by IMAGE/3000 in two ways:

- It serves as the default path for all accesses to the detail data set. That is, all backward and forward chain accesses are relative to the primary path until and unless the calling program explicitly switches to some other path.

- The restructuring utility, which copies a data base to tape, copies the entries of detail data sets as a series of primary path chains, one for each unique search item value. Reloading the data base from this tape results in the entries of each chain of the primary path being assigned contiguous storage locations. Subsequent chained accesses along the primary path exhibit time benefits similar to serial access.

The data base designer should designate the path most frequently employed in chained accesses as the primary path.

PRIVACY AND SECURITY

SECTION III

Privacy refers to those provisions which preclude unauthorized read access to the various elements of a data base. Security refers to those provisions which preclude unauthorized write access (including deletion) to the various elements of a data base.

3-1. EXTERNAL PRIVACY AND SECURITY

3-2. PRIVILEGED FILE PROTECTION

All files of IMAGE/3000 data bases are privileged files. Access by unprivileged processes or through most MPE/3000 file system commands is not allowed. This prevents accidental or deliberate privacy and security violations by non-privileged users. Privileged files are described in detail in the **Multiprogramming Executive Operating System** manual, (03000-90005).

Certain MPE/3000 commands pose threats to the privacy and security of IMAGE/3000 data bases. In particular the SYSDUMP and STORE commands permit System Supervisors, System Managers and other privileged users to copy files not currently open for output to tape. These and any other commands which permit copying of IMAGE/3000 files to tape, represent a potential breach of data base privacy, and their use should be controlled.

The MPE/3000 RESTORE command threatens data base security because it may purge and replace files of a data base with correspondingly named files encountered on tape. As with SYSDUMP and STORE, each data base manager should be informed whenever it is used.

3-3. ACCOUNT PROTECTION

Users of an IMAGE/3000 data base must be logged on the same account as the data base. This makes them subject to the control of the Account Manager and, hence, provides an account level of privacy and security.

3-4. GROUP PROTECTION

Users of an IMAGE/3000 data base must have write access to the group in which the files of the data base are catalogued. This makes them subject to the control of the Account Manager thus providing a group level of privacy.

3-5. INTERNAL PRIVACY AND SECURITY

3-6. SECURITY PROVIDED BY THE IMAGE/3000 UTILITIES

In contrast to those MPE/3000 commands which pose threats to data base security, the IMAGE/3000 utilities perform various checks to ensure data base integrity.

- They acquire exclusive access to the data base being processed.
- They execute only on behalf of the data base creator or, if specified by the creator, for any user supplying the correct maintenance word (see below) and only for users logged on 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.

When a data base is created, the creator may elect to specify an ASCII string, up to eight characters for the purpose of authorizing other users to apply IMAGE/3000 utilities to the data base. This string, called a **maintenance word**, resides in the root file. Users other than the creator may apply an IMAGE/3000 utility to the data base only if they supply this maintenance word.

3-7. READ LEVELS AND WRITE LEVELS

Within a data base definition, the data base designer is permitted to define privacy and security (read and write) levels for each data set and data item. A description of their specific application is included later in this section. In general, however, the following rules determine their use:

- specified read and write levels must be between 0 and 15
- default read levels and write levels are 0 and 15 respectively
- no read level can exceed the write level of the same element
- a low read level weakens the privacy of a data element
- a low write level weakens the security of a data element

3-8. LEVEL WORDS

The data base designer is permitted to specify up to 15 **level words** within the definition of a data base. Each level word is uniquely identified by one of the integers 1, 2, 3, . . ., 15, which are known as **level numbers**.

3-9. ACCESS LEVELS

Each user who opens a data base must supply, as one of the calling parameters, an ASCII string, which determines his access level. If no level words have been defined for the data base, the string is ignored and the user is assigned an **access level** of 15. If the string is found to match one of the level words defined for the data base, the user is granted an access level equaling the level number of the matching level word. If no match is found, he is assigned an access level one less than the lowest level number for which a level word was defined. In general, the higher the assigned access level, the greater his capability.

3-10. DATA ACCESS

Having been assigned an access level as described above, the ability of a user to access the entries and data items of each data set is prescribed as follows:

1. If his access level is less than the read level of a data set, he is not permitted to access the data set regardless of the read or write levels of the data items.

2. If his access level is greater than or equal to the read level of the data set, but is less than the write level of the data set, he is permitted **conditional** access to the data set, subject to the following:

   - he may read any data item, if his access level is not less than the read level of the data item
   - he may update any data item, except search or sort items, if his access level is greater than or equal to the write level of the data item
   - he is **not** permitted to add and delete data entries.

3. If his access level is greater than or equal to the data set write level, he is permitted **unconditional** access to the data set as follows:

   - he may read any data item
   - he may update any data item, except search or sort items
   - he may add and delete entries

3-11. LIBRARY PROCEDURE PRIVACY AND SECURITY PROVISIONS

1. All access to a data base is achieved through a Data Base Control Block (DBCB) resident in a privileged data segment not directly accessible to the user. His inability to read or modify the DBCB enables IMAGE/3000 to guarantee both the privacy and security of the data base.

2. All IMAGE/3000 library procedures which structurally modify the data base run with external interrupts inhibited. This prevents job termination while modifications are in progress. If any file system failures occur during such data base modification, IMAGE/3000 causes job termination since the data base integrity is suspect.

3. All buffers resident in the DBCB 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 user's program. This guarantees data base integrity despite any program termination which might occur between successive procedure calls.

DEFINING DATA BASES

SECTION IV

Data base definition is accomplished through the use of the Schema Processor. The Schema Processor reads a description of the user data base and produces an internal operational translation of it. This is deposited in a permanent file, called the root file. It contains descriptions of the relationships between data items, data sets, security levels and other parameters needed for subsequent data base access.

Generating a data base consists of three steps as shown in Figure 4-1:

a. Designing the data base.

b. Describing the design, using the data base description language, to create a schema.

c. Processing the schema through the Schema Processor to create the internal data base description (as contained in the root file).

4-1. DATA BASE DESCRIPTION LANGUAGE

The data base description language is used to define a data base. The data base description called a schema, may exist in the MPE/3000 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 user convenience (i.e., comments or collating information). The data base description language is a free-format language; the user can insert blanks anywhere in the schema to improve its appearance except within symbolic names and reserved words.

4-2. LANGUAGE CONVENTIONS

The following conventions are used in describing the data base description language:

1. Words in upper case and all punctuation appearing in format statements must appear exactly as shown. For example:

   SETS:

2. Replace lower case, bold-face words with user-supplied values. For example, replace **path count** with an integer between 0 and 16.

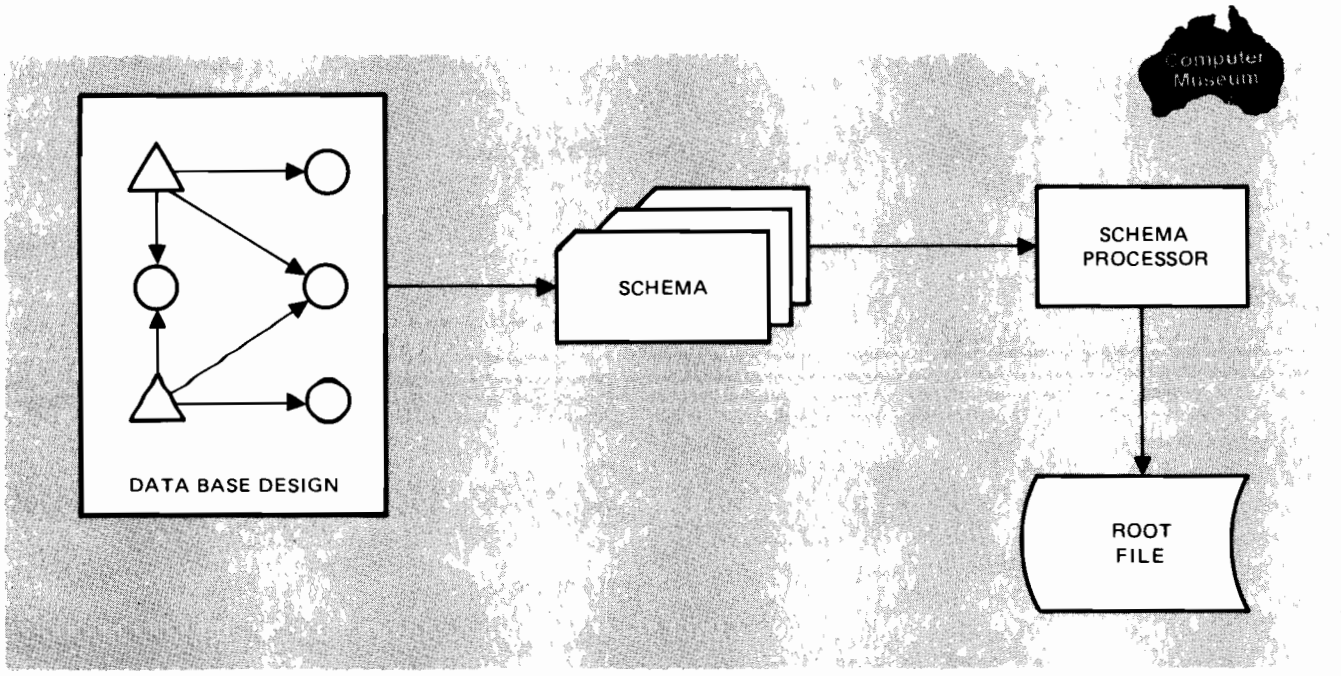

Figure 4-1. Data Base Design Process

3. Information in square brackets [ ] is optional. For example:

   item name [(path count)]
   means either
   item name (path count)
   or
   item name

4. Braces { } indicates that one piece of information within the braces must be specified. For example:

   NAME:**setname,** {DETAIL / D}
   means either
   NAME:**setname**,DETAIL;
   or
   NAME:**setname**,D

5. Comments take the form:

   <<**comment**>>

   Comments may be composed of any characters and may appear anywhere in a schema except imbedded in another comment. Comments are included in the schema listing but are otherwise ignored by the Schema Processor program.

6. All alphabetic input to the Schema Processor is upshifted, (converted to upper-case) with the exception of level words, which may contain lower case characters.

7. Data names may consist of from one to 16 alphanumeric characters, the first of which must be alphabetic. Any characters after the first must be chosen from the set: letters A-Z; digits 0-9; +; -; *; /; ?; '; #; %; &; .

4-3. SCHEMA STRUCTURE

The overall schema structure is:

BEGIN DATA BASE **data base name;**
LEVELS: **[level part]**
ITEMS: **item part**
SETS: **set part**
END.

data base name
is an alphanumeric string of from one to six characters. The first character must be alphabetic.

4-4. LEVEL PART DEFINITION. The **level part** provides the means for the designer to protect data items or data sets from unauthorized access. First he defines, in the **level part**, access level numbers and corresponding access level words; then he uses the level numbers when defining data items and data sets. As a result a user must furnish a proper access level word in order to access data associated with the corresponding access level number. See the discussion of privacy and security in Section III.

The format of the level part is:

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

level number
is an integer between 1 and 15, inclusive. Level numbers must be unique within the **level part** and must appear in ascending order.

level word
may consist of from one to eight ASCII characters (including lower case), except carriage return, semicolon, and blank. (Blanks will be compressed out by the Schema Processor.)

Below is an example of a **level part.**

```
LEVELS:
1   CLERK;
4   ;
8   MANAGER;
13  VP;
14  VP-EXEC;
15  PRESIDNT;
```

4-5. ITEM PART DEFINITION. Each data item in the data base must be defined in the **item part.** (This definition does not include the data set(s) in which an item appears. See **set part definition.**) A data item must be an integral number of words in length, not exceeding 2047, and 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. In this case, the data item is unique to the data set in which it appears.

Each data item definition is of the form:

item name, [sub-item count] type designator [sub-item length] [(read level, write level)];

item name
is an IMAGE/3000 data name as described earlier. It must be unique within the **item part.**

sub-item count
is an integer from 1 to 255, which denotes the number of occurrences of a sub-item within an item. If omitted it is implied to equal one. A data item whose **sub-item count** is 1 is a simple item. Otherwise it is a compound item.

type designator
denotes the form in which a sub-item value is represented in the computer. IMAGE/3000 does no conversion or checking, but allocates space for an item based on its **type designator, sub-item count,** and **sub-item length.**

There are three classes of **type designator**: word, byte and 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).

word designators

I
denotes a signed binary integer in 2's complement form.

J
denotes a signed binary number in 2's complement form which conforms to the specifications for COBOL/3000 COMPUTATIONAL data. (See **COBOL/3000, 03000-90014.**)

K
denotes an absolute binary quantity such as type LOGICAL in SPL/3000.

R
denotes a REAL (floating point) number such as those of FORTRAN/3000 and SPL/3000.

character designators

U
denotes an ASCII character string containing no lower case alphabetic characters.

X
denotes an unrestricted ASCII character string.

Z
denotes a zoned decimal format number compatible with COBOL/3000. (See **COBOL/3000, 03000-90014.**)

nibble designator

P
denotes a packed decimal number which conforms to the specifications for COBOL/3000 COMPUTATIONAL-3 data. (See **COBOL/3000, 03000-90014.**)

Table 4-1. Example of an **Item Part**

ITEMS:	
A,I2;	<< 32 BIT SIGNED INTEGER >>
MELVIN,3I (4,4);	<< COMPOUND ITEM. THREE SINGLE WORD SIGNED INTEGERS. READ LEVEL IS 4; WRITE LEVEL IS 4. >>
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) NUMBERS. READ LEVEL IS 1, WRITE LEVEL IS 8. >>
PURCHASE-MONTH, U8;	<< EIGHT-CHARACTER ASCII STRING WITH NO LOWER CASE ALPHABETICS.>>
MASK, K2;	<< 32 BIT ABSOLUTE BINARY QUANTITY.>>
TEMPERATURE, 17R3;	<< COMPOUND ITEM. 17 TRIPLE 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.>>

Table 4-2. Set Part for Master Data Sets

```
{NAME:}  set name,  {MANUAL   }                    [(read level, write level)]
{N:   }             {M        }
                    {AUTOMATIC}
                    {A        }

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

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

set name is an IMAGE/3000 data name

MANUAL (or M) denotes a manual master data set. Each entry within a manual master may contain one or more data items. These entries are created manually (i.e., the user must explicitly write them into the data set).

AUTOMATIC (or A) denotes an automatic master data set. Each data entry of an automatic master contains one data item only. IMAGE/3000 creates these entries automatically as needed.

sub-item length
is an integer from one to 255. It is the number of words, characters, or nibbles (depending on the **type designator** of the item) in a sub-item. If omitted it is implied to equal one. Each value of a data item 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 is denoted by the **type designator.** However, regardless of the **type designator,** each data item's length must equal an integral number of words. The entire item is always handled as a unit by IMAGE/3000.

(read level, write level)
is a pair of integers between 0 and 15, inclusive, which denote the data item access security levels. The **read level** number cannot exceed the **write level** number. If omitted, the schema processor assigns a **read level** of zero and a **write level** of 15. See Section III for an explanation of the functioning of read and write levels.

An example of an **item part** is given in Table 4-1 (located on the previous page).

4-6. **SET PART DEFINITION.** The **set part** of the schema names the various data sets within the data base, indicates which data items listed in the item part of the schema belong to which sets, and links the master data sets to the detail data sets by describing which of the data items in the data sets are search items. Each data set must have a unique name. No more than 99 data sets can be described in the set part, and each data set can contain no more than 127 data items.

Detail data sets must reference only those master data sets that have been previously defined in the schema **set part.**

The **set part** form for each master data set is given in Table 4-2.

A manual master entry must be added by the user **before** adding any detail entries with a matching search item value. An automatic master entry is added automatically if necessary when a corresponding detail entry is added.

read level, write level
is a pair of integers between 0 and 15, inclusive, which denote the data set access security levels. The **read level** number cannot exceed the **write level** number. If omitted, the Schema Processor assigns a **read level** of zero and a **write level** of 15. (See Section III for a complete explanation of levels.)

item name
is the name of a data item defined in the **item part.** A search item (see **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 to indicate the number of

Table 4-3. Set Part for Detail Data Sets

```
{NAME:}          set name,    {DETAIL}    [(read level, write level)];
{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:       }
```

set name is an IMAGE/3000 **data name.**

DETAIL or D denotes a detail data set.

paths (see discussion of paths in Section I), which will be established with various detail data sets. A zero **path count** (allowed with manual masters only) designates a master which is not linked to any detail data set. The zero **path count** merely denotes which item in the manual master is to be used as the search item for that data set. A path count must be specified for one, and only one, item in the master set.

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

The following is an example of two master data set definitions as they might appear in the set part of a schema.

```
NAME:SCHOOL-SPECS,MANUAL(9,15);
ENTRY:SCHOOL-ID(0),
      SCHOOL-NAME,
      SEMESTER,
      NO-SEMS,
      MODS-PER-DAY,
      DAYS-PER-CYCLE,
      T-OPTS,
      S-OPTS,
      R-OPTS;
CAPACITY:200;

NAME:TEACH-MSTR,A(8,14);
ENTRY:SCHL-TEACH(5);
CAPACITY:7000;
```

In the above example, SCHOOL-SPECS is a manual master data set with a read level of 9 and a write level of 15. The key item is defined as SCHOOL-ID. SCHOOL-ID has a path count of zero which indicates that the master is a stand-alone data set and is not linked to any detail data sets. The capacity of the data set is 200 entries.

TEACH-MSTR is an automatic master data set with a read level of 8 and a write level of 14. The only data item defined for the set is SCHL-TEACH, which is also defined as the key item. SCHL-TEACH has a path count of 5 which indicates that the master has five links to data sets. The capacity of the data set is 7000 entries.

The **set part** form for each detail data set is given in Table 4-3.

read level, write level
is a pair of integers between 0 and 15, inclusive which denote the data set access security levels. The **read level** number cannot exceed the **write level** number. If omitted, the Schema Processor program assigns a **read level** of zero and a **write level** of 15. (See Section III for an explanation of levels.)

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

!
denotes a primary path. Only one path can be so designated for each data set. If no path is designated as primary by the user, the first unsorted path (i.e., the first item linked to a master and which does not have a sort item), is the primary path by default. If all of the paths are sorted, then 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 the item name in a detail data set, this indicates that the data item is a search item. The master set name indicates which master data set is linked to the detail set. Up to 16 such search items can be defined for a detail data set. If none of the data items have a master set name following, the detail data set is not related to any master data set. Since the same data item name may appear in more than one data set, it is possible that the search item in a detail data set and the search item of a master may have the same description as defined in the schema.

If a detail data set is not linked to any master data set, then 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 located in the entry currently being defined. Specifying a sort item defines a sorted path. Each entry added to a chain of a sorted path will be linked logically in order of the ascending values of the sort item. If sort item name is omitted, the path is chronoligical, and new entries are logically linked to the ends of member chains.

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

Following is an example of a detail data set definition.

```
NAME:TEACH-CENSUS,DETAIL(6,12);
ENTRY:SCHL-TEACH(TEACH-MSTR),
        FIRST-NAME,
        LAST-NAME,
        BIRTH-PLACE,
        BIRTH-DATE,
        SEX,
        SALARY-GROUP(!SALARIES(LAST-NAME)),
        CIT-CODE
        SCHOOL-ADR,
        PHONE-NO;
CAPACITY:1000;
```

TEACH-CENSUS is linked to TEACH-MSTR, a previously defined master data set, by a search item name SCHL-TEACH. The description of SCHL-TEACH in the item part is identical to the description of the key item for TEACH-MSTR.

SALARY-GROUP is the search item of the primary path of this data set. It is linked to a master called SALARIES and the detail entries of SALARY-GROUP are chained in LAST-NAME order.

TEACH-CENSUS can hold 1000 entries.

4-7. SCHEMA PROCESSOR OPERATION

4-8. OPERATING INSTRUCTIONS

The Schema Processor executes in either MPE/3000 Job or Session mode. For further information, consult **HP 3000 Multiprogramming Executive Operating System, 03000-90005.**

To execute the Schema Processor, the user enters the MPE command:

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

The Schema Processor scans a schema and if no errors are detected (and the root file option is not negated), produces a root file.

In addition to the root file, two files are of significance to the Schema Processor. The schema input is obtained from a **textfile** and output is listed on a **listfile.** Each of these files is referenced in the Schema Processor by a formal file designator. Either or both of these may be equated by the user to actual file designators prior to executing the Schema Processor. A default actual file designator exists for each, as specified in Table 4-4.

The :FILE command can be used to equate an actual file designator to a formal file designator. (See **HP 3000 Multiprogramming Executive Operating System, 03000-90005** for further details.) If a :FILE command is included in the job stream, the user 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 4-5 shows examples of the RUN command.

Table 4-4. Schema Processor Files

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

Table 4-5. RUN Comands

:RUN DBSCHEMA.PUB.SYS	Uses all default files
: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 read and printed by the Schema Processor. If the schema is error-free and the user so requests (see CONTROL COMMAND), a root file is created whose name is the same as the data base name specified in the schema. The user for whom the root file is created can subsequently create and initialize the data base by executing the utility program DBUTIL (see Section VI).

4-9. SCHEMA PROCESSOR COMMANDS

The Schema Processor subsystem commands are entered from the **textfile**. (See Figure 4-2.)

The basic syntax of subsystem commands is:

$command [**parameter-list**]

The $ must be the first character of the record, immediately followed by the command name, which must be completely spelled out. The **parameter-list**, separated from the name by at least one blank, is optional; some commands have parameters -- others do not. Parameters are separated from each other by commas. Blanks may be freely inserted between items in the list.

A command is continued to the next record if the last non-blank character is an ampersand (&). In this case, the following record must begin with a $. The effect is to combine the characters preceeding the & with those following the $ of the next record with a blank inserted between them. Therefore, command names and other elements cannot be broken by &. Characters beyond the 72nd character of each record are ignored.

Schema Processor command records may occur anywhere in the schema. They may not contain comments.

4-10. PAGE Command. The form for this command is:

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

The **PAGE** command causes the **listfile** to eject to the top of the next page, where the optional character-strings are printed and two more lines are skipped. Listing continues at that point. The strings must be enclosed in quotes (which are deleted, and the strings concatenated) and separated by commas. A quote mark is specified in the **character-string** by a pair of quotes. The PAGE command is effective only if the LIST option (see CONTROL command) is on. The PAGE command itself is not sent to

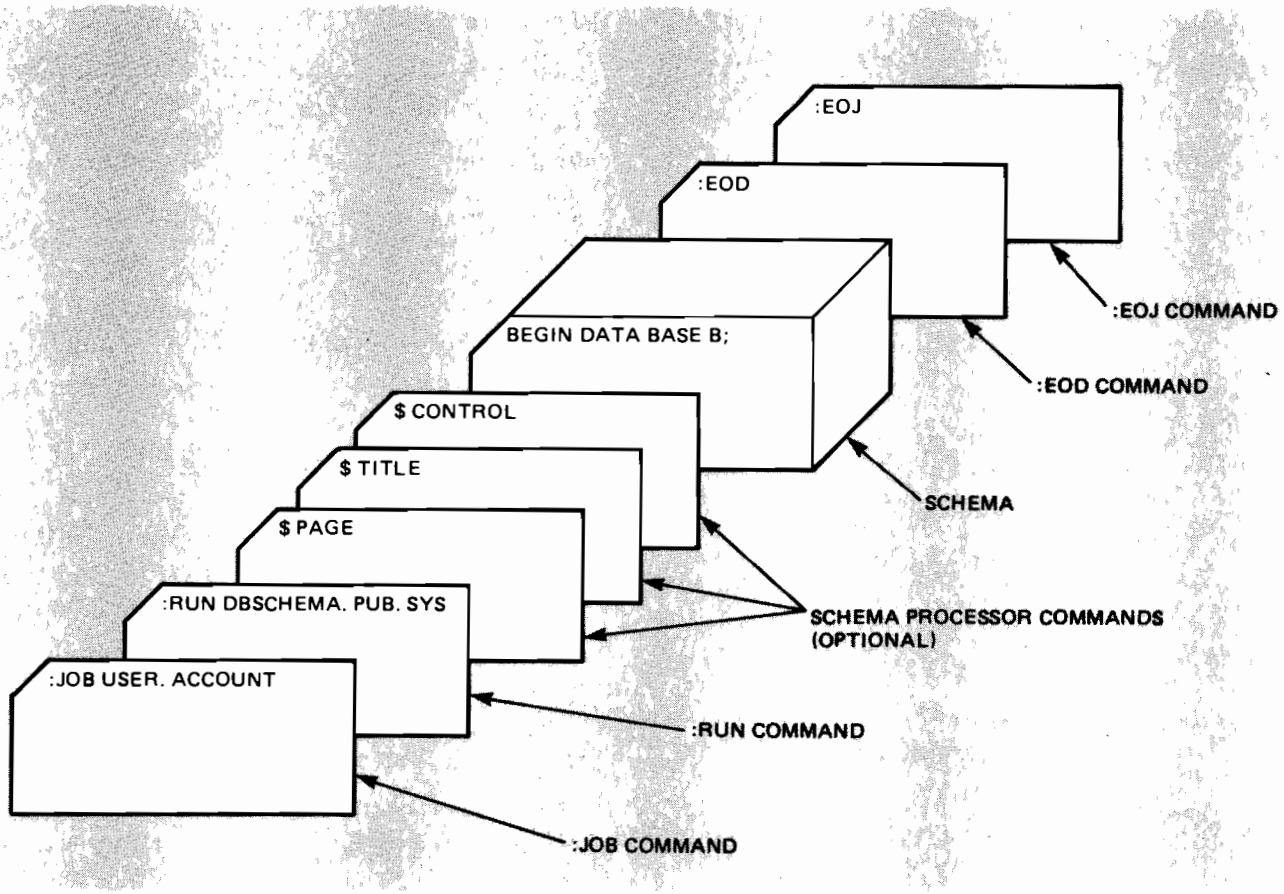

Figure 4-2. Schema Processor Batch Job Stream

the listfile. The **character-string** replaces that specified by a previous TITLE command. If **"character-string"** is omitted, the preceding TITLE (if any) is printed at the top of the next page.

Examples:

```
$PAGE "MASTER DATA SETS"
$PAGE
```

4-11. **TITLE Command.** The form for this command is:

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

The **character-strings** are stripped of quotes, concatenated and saved. The resulting string is used whenever a page heading is printed (unless overridden by a PAGE command or a new TITLE command). A quote mark is specified in a **character string** by a pair of quotes. If no **character string** is specified no title will be printed from that point on.

4-12. **CONTROL Command.** The form for this command is:

```
$CONTROL [[parameter],. . .]
```

The possible parameters of the CONTROL command are described in Table 4-6.

The default parameters for the $CONTROL command are:

```
LIST
ERRORS = 100
LINES = 60 if listfile is a line printer
      = 32767 otherwise
ROOT
BLOCKMAX = 512
TABLE
```

Table 4-6. CONTROL Command Parameters

LIST	Causes each source record to be printed on the listfile.
NOLIST	Only source records with errors are printed on the listfile, followed by the error message.
ERRORS=**nnn**	Sets the maximum number of errors to **nnn**. (0≤**nnn**≤999). If more than **nnn** errors are detected, the Schema Processor stops.
LINES=**nnnnn**	Sets the number of lines per page on the listfile to **nnnnn** (4≤**nnnnn**≤32767).
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 each data set. (128≤**nnnn**≤2048). This constrains the number of media records which will be combined in each physical block.
TABLE	Causes the Schema Processor, if no errors are detected, to write a table of summary information about the data sets to the listfile device.
NOTABLE	Suppresses TABLE.

4-13. SCHEMA PROCESSOR OUTPUT

The Schema Processor prints a heading on the user **listfile** which includes a page number, product identification, product name (IMAGE/3000), and the date and time. If the user's standard output device (\$STDLIST), is different from his **listfile**, an abbreviated product identification is also output to \$STDLIST. Subsequent pages of **listfile** are headed by a page number, the data base name (if yet encountered), and the title most recently specified by a TITLE or PAGE command (if any).

If the user is entering the schema from an interactive terminal, the Schema Processor prompts with a ">" when it is ready for a new line of input.

If the **textfile** and **listfile** are distinct, and if the LIST option is active, a copy of each record of the schema is sent to the **listfile**.

After the entire schema has been scanned, several types of summary information may be output to 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: followed by a list of items defined but not referenced in a data set will be written to the **listfile**. 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. (See Figure 4-3 for sample printout.)

The information printed is described in Table 4-7.

The NOTABLE parameter of the CONTROL command suppresses printing of the table.

Two lines of summary totals are printed on the **listfile**. An example of these is:

```
NUMBER OF ERROR MESSAGES:0
HIGHEST LEVEL WORD: 15
ITEM NAME COUNT: 22
DATA SET COUNT: 6
```

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

If no schema syntax or logical errors were encountered, a third line is also printed. It takes the form

```
ROOT LENGTH: x  BUFFER LENGTH: y
TRAILER LENGTH: z
```

Table 4-7. 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
LEVEL R W	The data set read level number appears under R and the data set write level number appears under W.
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.
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.
DISC SPACE	The amount of disc space (in 128-word sectors) occupied by the MPE/3000 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.

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

EMPLOYEE            M   3  14   4    1   7     17   500       30   512   72
PROJECT-MASTER      M   3  14   2    1   10    20   75        19   382   15
LABOR               D   3  9    4    2   10    18   10024     28   506   1436
                                  TOTAL DISC SECTORS INCLUDING ROOT: 1528
```

Figure 4-3. Data Set Summary Table

These three quantities are in words. ROOT LENGTH is the length of the body of the root file. BUFFER LENGTH is the length of each of the four buffers which IMAGE/3000 will allocate in an extra data segment for use in transferring data set blocks to and from disc. TRAILER LENGTH is the length of an area in the extra data segment used by IMAGE/3000 to transfer information to and from a calling program's stack. An approximation of the size of the necessary extra data segment is **x** + 4**y** + **z** words.

If the ROOT option is active (and if no schema errors were detected) then the Schema Processor creates the root file, initializes and saves it as a catalogued MPE/3000 disc file.

4-14. SCHEMA ERRORS

If the LIST option is active, and an error occurs, the Schema Processor sends an error message to the **listfile** immediately following the offending statement. If NOLIST is active, and an error is detected, the Schema Procesor first prints the current line of the schema and then the error message. Consult **Appendix B, Schema Processor Errors** for an explanation of errors generated by Schema Processor commands. Any of the errors listed will prohibit creation of the root file. However, the Schema Processor will attempt to continue checking the schema for logical and syntactical correctness. In some cases one error (particularly if it occurs early in a data set), obscures detection of subsequent errors. Additional execution(s) of the Schema Processor are then needed to reveal the subsequent error(s). Conversely, some errors early in the schema can generate subsequent apparent errors which will disappear after the original error has been corrected. Consult **Multiprogramming Executive Operating System, 03000-90005** for errors generated from the use of MPE/3000 commands.

The following message is sent to the **listfile** (and to $STDLIST, if different) if no errors are detected and if the ROOT option is active:

ROOT FILE **data base name** CREATED.

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

If schema errors are detected which prohibit creation of the root file, the following message is sent to the **listfile** (and to $STLIST, if distinct):

PRECEDING ERRORS -- NO ROOT FILE CREATED.

A few conditions, such as 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 output.

SCHEMA PROCESSING TERMINATED

ACCESSING DATA BASES

SECTION V

Within a user's program, accessing IMAGE/3000 data bases is accomplished by calls to the IMAGE/3000 library procedures. These procedures provide the following functions:

- Initiate user access (open a data base)
- Read and update data items
- Read, write, and delete entries
- Report name, structure, and organization information
- Terminate user access (close a data base)

Accessing a data base involves the following steps:

- Opening a data base
- Performing all data accesses
- Closing the data base

5-1. ACCESS MODES

When a user initiates access to a data base, he specifies one of three possible modes. The mode selected, known as the **access mode,** determines the types of operations that he can subsequently perform on the data base. Access modes are mutually exclusive in that all concurrent users must open the data base in a common mode.

- In access mode 1, each user must gain temporary exclusive control of the data base prior to reading, updating, adding, or deleting any entries. He must subsequently relinquish this exclusive control to permit other access mode 1 users to access the data base. Acquiring and relinquishing is referred to as **locking** and **unlocking**, respectively, and both functions are supplied by IMAGE/3000 library procedures.

- In access mode 2, users are not permitted to add or delete entries. However, they are permitted to read and update entries. Attempts to lock or unlock the data base are ignored.

- In access mode 3, only one user has access to the data base. He may read, update, add or delete entries. Attempts to lock, or unlock, the data base are ignored.

Some special considerations apply to mode 1 and mode 2 users.

In access mode 1:

- Failure of one process to unlock (or close) the data base will cause all other processes attempting to lock the data base to remain suspended until the offending process terminates.

- Unlocking results in a **logical** closing of all data sets. This is necessary since positional information relative to data sets is meaningless in light of the fact that other processes may perform extensive deletions and additions on the data base before the given process regains control.

- Locking and unlocking a data base involves acquiring and releasing exclusive use of an MPE/3000 global **resource identification number** (RIN). Therefore, locking data bases is subject to the same constraints as locking files or explicitly locking RINs (see **HP 3000 Multiprogramming Executive Operating System, 03000-90005**). Specifically, this means that a user who desires to lock simultaneously two or more data bases, or other resources, must have multiple RIN capability. However, users with this capability must be careful to avoid deadlocking, in which two or more suspended processes cannot be resumed because some of them are mutually blocked.

In access mode 2:

- If two or more processes update data items of different entries, or different data items of the same entry, the data base will correctly reflect all changes.

- If two or more processes update the same data items of the same entry, the data base will reflect only the latest values.

- A read of an entry may, or may not, reflect the latest update by other users depending on whether the latest update was done after or before the buffer (in the DBCB) containing the entry was loaded from disc.

Table 5-1. IMAGE/3000 Call Statements

Host Language	Call Statement
SPL	name (parameter, parameter, . . ., parameter)
COBOL	CALL "name" USING parameter, parameter, . . ., parameter
FORTRAN	CALL name (parameter, parameter, . . ., parameter)

5-2. PROCEDURE CALLS

All library procedures may be called directly from user programs, regardless of the host language. This is possible since none of them are TYPE procedures, none of them utilize the OPTIONS VARIABLE capability of SPL, and all parameters are call-by-reference word pointers.

The forms of the call statements for various host languages are shown in Table 5-1. In the table, **name** refers to a procedure name and **parameter** to one of its parameters.

If the host language is SPL, all procedures called must be declared as EXTERNAL procedures or, alternatively, must be named in an INTRINSIC statement. This is not required in other host languages.

5-3. PROCEDURE CALL ERRORS

All IMAGE/3000 library procedures alter the HP 3000 hardware condition code as represented in the Status register. Any checks of the condition code should be made immediately upon return from the procedure.

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.

In addition to the condition code, all procedures also return from 1 to 10 words of status information to a user supplied status buffer which is a required parameter for all procedure calls. The first word of this status area is called a **condition word** and its values correspond to the condition code as follows:

Condition	Condition Word
CCE	= 0
CCG	>0
CCL	<0

Host languages may check the condition word, instead of the condition code, to determine the success or failure of the procedure. Moreover, various positive and negative condition words are employed to indicate the nature of exceptional conditions and errors. These are described in Appendix C, **Library Procedure Errors.**

Under certain conditions, the calling process may be terminated by IMAGE/3000. Conditions giving rise to process termination and a description of the accompanying error messages are presented in Appendix C.

5-4. LIBRARY PROCEDURE PARAMETERS

All library procedures are described in this section. A sample data base and programs employing the procedures are provided in Appendix A.

An SPL procedure declaration head is shown with the discussion of each procedure. The SPL procedure call format is enclosed in a box as follows:

PROCEDURE | procedure name (parameter list); |

specification part

OPTION CHECK 2,EXTERNAL;

The **specification part** includes the type and name of each parameter of the **parameter list.** The CHECK 2 option precludes calling any IMAGE/3000 library procedure with the wrong number of parameters or as a typed procedure. In either case a program load error will occur.

Each library procedure is invoked with up to seven call-by-reference parameters. The formal name, format, purpose and general interpretations of each parameter is described below:

BASE
This parameter identifies the data base being accessed.

When opening the data base, the first word of BASE must consist of a pair of ASCII blanks. The remaining bytes identify the data base by means of a left-justified name terminated by a semicolon or blank. To access a data base catalogued in a group other than the user's log-on group, the name must be a qualified name consisting of a data base name followed by a period followed by the group name (i.e., COMPNY.GROUP7).

A successful opening of the data base results in the pair of blanks in the first word of BASE being replaced by the extra data segment number of the assigned Data Base Control Block. In all subsequent accesses the first word of BASE must be this extra data segment number.

LEVELWORD
This word array contains a left-justified ASCII string, either eight characters long or, if shorter, terminated by a semicolon or blank.

This parameter determines the access level assigned to the user as a result of opening the data base.

DSET
This parameter is used to identify a data set.

It may contain a left-justified data set name, either 16 characters long or, if shorter, terminated by a semicolon or a blank.

Alternatively, it may be an integer referencing the data set by number as determined by the order of its appearance in the schema. The first data set in the schema is number 1, the second number 2, and so on.

MODE
This parameter is used to qualify the mode of operation of a procedure.

In all cases it must be an integer, and varies in value depending upon the library procedure being called. The value of MODE in one procedure call has no relation to the value of MODE in any previous or subsequent procedure call.

STATUS
This array is used as a communication area in which up to 10 words of control (status) information is returned to the user.

ITEM
This parameter is used to identify a data item.

It may contain a left-justified data item name, either 16 characters long or, if shorter, terminated by a semicolon or a blank.

Alternatively, it may be an integer referencing the data item by number as determined by the order of its appearance in the schema. The first item in the schema is number 1, the second, number 2, and so on.

LIST
This parameter is used to reference an ordered set of data items, either by name or by number, within an entry.

It may contain a left-justified set of data item names, separated by commas and terminated by a semicolon or blank. No interior blanks are allowed and no name may appear twice.

When referencing by number, the first word of the LIST array is an integer **n** which is followed by **n** data item numbers, none of which may appear twice.

Some special constructs are permitted as follows:

- An "empty" LIST consisting of an ASCII zero followed by a semicolon or blank, or, simply, a semicolon or blank.
- An "empty" numeric LIST in which **n** = 0.
- An asterisk followed by a semicolon or blank (*;).
- A commercial at sign followed by a semicolon or blank (@;).

The empty LIST requests no data transfer.

The asterisk LIST indicates that the procedure will use the previous LIST applied to the same data set.

The commercial at sign LIST serves to reference all the data items of the data set in the order of their occurrence in the entry. It provides a "full-record-mode" of accessing data entries.

QUALIFIER
This parameter is used to reference a data item or data set, either by name or by number.

The form of this parameter is identical to DSET and ITEM as described above.

BUFFER
This array serves as the logical I/O area between the user and the data base. All data transferred to or from the user by the library procedure is put into or taken from the BUFFER area.

Within BUFFER, data items exist as concatenated values which are in the same order as their data item names (or numbers) in the corresponding LIST parameter.

ARGUMENT
In calculated retrieval, this array contains a single data item value.

In directed retrieval, it contains a double word record number.

5-5. DBOPEN

Before a user's process can access an IMAGE/3000 data base, it must initiate access to the data base with a DBOPEN procedure call.

If the data base is successfully opened, the pair of blanks preceding the data base name (see the description of BASE parameter) is replaced with a data segment number used in all subsequent accesses to this data base.

A process issuing more than one successful DBOPEN on the same data base, results in multiple, logically distinct, access paths to the data base. DBOPEN returns unique data segment numbers for each such access path. IMAGE/3000 also maintains separate sets of status information for each access path.

In opening a data base, DBOPEN establishes an access path between the data base and the user's program by:

- Verifying the user's right to access the data base under the security provisions provided by the MPE/3000 file system.
- Determining that the data base is not currently open in the exclusive mode or in a mode differing from that requested by the user.
- Constructing a Data Base Control Block as the access path required by the other IMAGE/3000 library procedures in accessing the data base. The information in the data base control block is derived from the root file but is modified slightly as influenced by the LEVELWORD and MODE parameters of the DBOPEN call.

5-6. DECLARATION

```
PROCEDURE DBOPEN (BASE, LEVELWORD,
                  MODE, STATUS);
ARRAY BASE, LEVELWORD, STATUS;
INTEGER MODE;
OPTION CHECK 2, EXTERNAL;
```

5-7. OPERATION

- The root file referenced by BASE is opened either with dynamic locking, without dynamic locking, or exclusively depending on the value of MODE. The root file remains open until the data base is closed. This prevents the data base from being opened in more than one MODE.
- An extra data segment is allocated and initialized as a Data Base Control Block. The initialization differs slightly depending on the values of LEVELWORD and MODE.
- The pair of blanks in the first word of BASE is replaced with the data segment number of the DBCB. All subsequent accesses are via this data segment number and the referenced DBCB.
- Three integers are returned in the STATUS array. They consist of a condition word of zero, the user's access level (0 to 15 as determined by the contents of LEVELWORD), and the word size of the DBCB.
- A condition code of CCE is returned to the user. A negative condition word and a condition code of CCL reflects an unsuccessful operation. The various possible negative condition words and their meanings are described in Appendix C.

The valid values of MODE are 1, 2 and 3 with the following meanings:

MODE = 1

The root file is opened for shared, read-write access with dynamic locking, which can succeed only if all other current users of the data base have access mode 1. A flag is turned off, indicating that the data base is disabled for data access. The data access procedures DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE are disabled by the state of this flag. The user must call DBLOCK to turn the flag on and enable data access and subsequently DBUNLOCK to turn it off again.

MODE = 2

The root file is opened for shared, read-write access without dynamic locking, which can succeed only if all other users, if any, have access mode 2. The data base is enabled and remains enabled for data access, except for DBPUT and DBDELETE which are permanently disabled.

MODE = 3

The root file is opened for esclusive read-write access, which can succeed only if there are currently no other users. The data base is enabled and remains enabled for any data access.

The value of LEVELWORD determines the access level granted the user as explained in Section III.

The DBCB tables reflect the users access mode and access level for proper handling of all subsequent procedure calls.

5-8. DBINFO

This procedure is employed to obtain information about any data base currently being accessed. The user may obtain information about data items, data sets, or paths. The information returned is restricted by the user's access level; any data items, data sets, or paths of the data base inaccessible to the user are considered to be non-existent.

5-9. DECLARATION

```
PROCEDURE  DBINFO (BASE, QUALIFIER,
             MODE, STATUS, BUFFER);
ARRAY BASE, QUALIFIER, STATUS, BUFFER;
INTEGER MODE;
OPTION CHECK 2, EXTERNAL;
```

5-10. OPERATION

- Information about the data base BASE is returned in the BUFFER. Its content is determined by the values of MODE and QUALIFIER as described below.
- The user's condition code is set to CCE and a condition word of 0 is returned in the first word of STATUS.

Various other condition words, corresponding to condition codes other than CCE, are described in Appendix C.

The 100, 200, and 300 series of MODE values are employed to obtain information about, respectively, data items, data sets, and paths as follows:

MODE = 101

QUALIFIER must reference a data item which is accessible to the user.

If the user can update the specified data time in at least one data set, or if he can add or delete entries containing it, a negative data item number is returned as the first word of BUFFER. Otherwise, a (positive) data item number is returned.

MODE = 102

QUALIFIER must reference a data item which is accessible to the user. Thirteen words of information, all pertaining to the referenced data item, are returned in the BUFFER as follows:

Words	Contents
1-8	The left-justified data item name, padded with blanks, if needed.
9	The data item type (I, J, K, R, U, X, Z or P) followed by a blank.
10-13	The sub-item length, sub-item count, read level and write level, all as integers.

MODE = 103

QUALIFIER is ignored.

An integer count of the number of data items accessible to the user is returned as the first word of BUFFER. This is followed by an array of signed data item numbers, in data ~~item number order,~~ identifying all data items accessible to the user. A positive number indicates that the user has read-only access to the data item. A negative number means he has update access to the item in at least one data set, or can add or delete entries containing the item.

MODE = 104

In this mode, QUALIFIER must reference a data set accessible to the user.

A word array, entirely similar to MODE 103, is returned to BUFFER. In this case however, the data item count and the array of signed data item numbers pertain to only those items accessible in the referenced data set. In this mode, the signed data item numbers appear in the order of their occurrence in the data entries.

MODE = 201

QUALIFIER must reference a data set accessible to the user.

The signed data set number of the referenced data set is returned as the first word of BUFFER. This number is negative if the user can add and delete entries of the data set; otherwise it is positive.

MODE = 202

QUALIFIER must reference a data set accessible to the user. Seventeen words of information, all pertaining to the referenced data set, are returned in the BUFFER as follows:

Words	Contents
1-8	The left-justified data set name, padded with blanks, if needed.
9	The data set type (M, A, or D), followed by a blank.
10,11	The integer word length and blocking factor of the data entries.
12,13	The integer read level and write level of the data set.
14,15	The doubleword number of entries currently in the set.
16,17	The doubleword integer capacity of the set.

MODE = 203

QUALIFIER is ignored.

An integer count of the number of data sets accessible to the user is returned as the first word of BUFFER. This is followed by an array of signed data set numbers, in data set number order, identifying each accessible data set. A negative number means the user can add and delete entries in the data set. A positive number means he may read, and possibly update, data items of existing entries.

MODE = 204

In this mode, QUALIFIER must reference a data item accessible to the user.

A word array, entirely similar to MODE 203, is returned to BUFFER. In this case, however, the data set count and array of signed data set numbers identify only those data sets in which the referenced data item is accessible.

MODE = 301

QUALIFIER must reference a data set accessible to the user.

An integer count of the number of paths (relationships) defined for the data set is placed in the first word of BUFFER. If this count is not zero, it is followed by a corresponding number of 3-word **path designators.**

If QUALIFIER references a master data set, each **path designator** consists of three integers identifying, in order, the data set number, search item number, and sort item number for the corresponding path of the related detail data set.

If QUALIFIER references a detail data set, each **path designator** consists of three integers identifying, in order, the related master data set number, and the defining detail search item number and sort item number for the path.

In each instance, path designators are presented in the order of their appearance within the schema.

A sort item number of zero is returned if no sort item was defined for the path or if the sort item is not accessible to the user. A path designator is not included if the user does not have access to its search item.

MODE = 302

QUALIFIER must reference a data set accessible to the user.

Two integers are returned to the BUFFER.

If QUALIFIER references a master data set, the first integer is the search item number and the second is a zero. If the search item is accessible, both integers are zero.

If QUALIFIER references a detail data set, the first integer is the search item number for the primary path and the second is the data set number of the related master data set. However, if the search item of the primary path is inaccessible, both integers are zero.

5-11. DBCLOSE

DBCLOSE is employed to terminate access to a data base. It may also be employed to terminate, temporarily or permanently, access to a data set.

5-12. DECLARATION

```
PROCEDURE  DBCLOSE (BASE, DSET, MODE,
                    STATUS);
ARRAY BASE, DSET, STATUS;
INTEGER MODE;
OPTION CHECK 2, EXTERNAL;
```

5-13. OPERATION

- The data set referenced by DSET, of the data base BASE is verified to be accessible to the user.
- All data sets and the root file of the data base referenced by BASE are closed.
- The extra data segment serving as a Data Base Control Block is released, and the first word of BASE is replaced with a pair of ASCII blanks.
- The condition code is set to CCE and the condition word is set to zero.

Various condition words corresponding to condition codes other than CCE are described in Appendix C.

The data set referenced by DSET must be accessible.

MODE = 1

Access to the data base is terminated as described above.

MODE = 2

In this mode, access to the data base is not terminated and only the data set referenced by DSET is closed. This mode may never be required. It is provided to permit the specified data set to be rewound and, perhaps more importantly, to close it when finished with it. Rewinding the data set resets the dynamic status information (in the DSCB) to the initial state (as if the data set were not yet accessed).

Each open data set, and file, requires system resources. Mode 2 enables these resources to be returned to the system without terminating access to the data base. The first subsequent access, if any, to a data set explicitly closed in this manner will be treated as if it were the first access to the data set since the data base was opened.

5-14. DBFIND

DBFIND is employed in calculated access to the entries of a chain of a detail data set. It locates the first and last entries of the chain and alters internal status information in anticipation of subsequent accesses to the members of the chain.

5-15. DECLARATION

```
PROCEDURE  DBFIND (BASE,DSET,MODE,
             STATUS,ITEM,ARGUMENT);
ARRAY BASE,DSET,STATUS,ITEM,
      ARGUMENT;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-16. OPERATION

- The detail data set referenced by DSET, of the data base BASE is verified to be accessible to the user.
- The internal tables for the referenced data set are examined to determine which path is defined by the search item referenced by ITEM.
- The related master data set is accessed (through calculated access) to obtain the chain head of the detail data set chain whose identifying search item value is ARGUMENT.
- The current values of chain count, backward pointer, and forward pointer as maintained internally for the detail data set DSET are replaced with the corresponding values within the matching chain head. The internal current path number is set to the new path and the current record number is set to zero.
- Ten words, reflecting the status of the data set DSET, are returned in the STATUS array as follows:

Words	Contents
1	A condition word of 0.
2	A zero (no data transferred).
3-4	A doubleword current record number of 0.
5-6	A doubleword chain length count.
7-8	A doubleword record number of the last entry in the chain.
9-10	A doubleword record number of the first entry in the chain.

- A condition code of CCE is returned to the user.

Various condition words resulting in condition codes of CCL or CCG, are described in Appendix C.

The value of MODE must always be 1.

5-17. DBGET

DBGET enables the user to read the data items of a specified entry. Eight different ways are provided for specifying the entry to be read; the actual reading is accomplished independent of the entry specification method.

5-18. DECLARATION

```
PROCEDURE  DBGET (BASE,DSET,MODE,
               STATUS,LIST,BUFFER,
                     ARGUMENT);
ARRAY BASE,DSET,STATUS,LIST,
      BUFFER,ARGUMENT;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-19. OPERATION

- The data set referenced by DSET, of the data base BASE, is verified to be accessible to the user.
- The internal tables for the referenced data set are used in conjunction with the value of MODE and/or ARGUMENT (see below) to determine the record number of the entry to read.
- The data item values of the data items specified by LIST, are extracted from the specified entry and placed in the BUFFER in the same left-to-right order as their occurrence in LIST.
- The chain pointers, determined by the current path, are used to update the internal backward and forward pointers for DSET. If DSET is a master data set, the chain pointers are synonym chain pointers. If DSET is a detail data set with at least one path, the current path is determined by the last successful DBFIND, unless no DBFIND was employed. In the latter case, the current path is the primary path. If the data set has no paths, these pointers are zeroes.
- The current address and data length fields are updated to reflect the record address of the data entry being read and the number of words transferred to BUFFER. The internal chain length count is set to zero, except when reading a primary entry of a master data set, in which case it is set to the synonym chain count.
- Ten words, reflecting the new internal status for the data set DSET, are placed in the user's STATUS array as follows:

Words	Contents
1	A condition word of zero.
2	An integer representing the word length of the logical entry read into BUFFER.
3-4	The doubleword record number of the data entry involved in the data transfer.
5-6	A doubleword zero, unless the entry read is a primary entry, in which case it is the synonym chain count.
7-8	The doubleword record number of the predecessor entry in this chain of the current path.
9-10	The doubleword record number of the successor entry in this chain of the current path.

- A condition code of CCE is returned to the user.

Various condition words corresponding to condition codes of CCL and CCG, are described in Appendix C.

The LIST must reference data items accessible to the user in the entries of the data set referenced by DSET, and no data item may be referenced twice.

An "empty" LIST, consisting of an ASCII zero or empty (blank or semicolon) or an integer 0, can always be employed. In this case no data transfer occurs.

A special LIST consisting of an asterisk (*;) indicates that the data items involved in the transfer are the same as the last LIST previously applied to this data set in an IMAGE/3000 library procedure call.

A special LIST consisting of a commercial at sign (@;) indicates "full-record-mode" access in which the entire data entry is transferred to the BUFFER array. It is applicable only to data sets with write levels not greater than the user's access level.

The different ways of specifying the entry to be read correspond to eight different values of MODE as follows:

MODE = 1, Re-read

Read the entry at the internally maintained current record address. ARGUMENT is ignored.

MODE = 2, Serial Read

Read the first entry whose record address is greater than the internally maintained current address. ARGUMENT is ignored.

MODE = 3, Backward Serial Read

Read the first entry whose record address is less than the internally maintained current address. ARGUMENT is ignored.

MODE = 4, Directed Read

ARGUMENT is treated as a doubleword record number. Read the entry (if it exists) at the corresponding record address.

MODE = 5, Chained Read

Read the entry referenced by the internally maintained forward pointer. ARGUMENT is ignored.

MODE = 6, Backward Chained Read

Read the entry referenced by the internally maintained backward pointer. ARGUMENT is ignored.

MODE = 7, Calculated Read

ARGUMENT is treated as a key value for the master data set referenced by DSET. Read the entry with matching key value.

MODE = 8, Primary Calculated Read

ARGUMENT is treated as a key value for the master data set referenced by DSET. Read the entry occupying the primary address of ARGUMENT if the entry is a primary entry.

5-20. DBUPDATE

This procedure permits the user to modify entries of data sets to which the user has at least read access. The user must have write access to each data item whose value will actually change. Note that search item and sort item values cannot be changed.

5-21. DECLARATION

```
PROCEDURE  DBUPDATE (BASE,DSET,MODE,
                     STATUS,LIST,BUFFER);
ARRAY BASE,DSET,STATUS,LIST,BUFFER;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-22. OPERATION

- For the specified BASE, the entry residing at the current record address of the data set referenced by DSET is modified.

- The BUFFER array is considered as a concatenated set of data item values in the same order, left-to-right, as the items referenced by LIST. These values are used to replace the corresponding items of the entry.

- The LIST may reference data items which the user is not permitted to alter providing that the corresponding values in the BUFFER are the same as the current values of the data items. This permits reading and updating with the same LIST.

- The data length field is updated to reflect the size of BUFFER.

- Ten words, reflecting the status of the data set DSET, are placed in the STATUS array as follows:

Words	Contents
1	A condition word of zero.
2	The word length of the logical entry in BUFFER.
3-10	The same doubleword values of current record number, chain count, predecessor record number and successor record number as returned by the preceding procedure call which positioned the data set at the current entry.

- A condition code of CCE is returned to the user.

Various condition words corresponding to the condition codes of CCL and CCG are discussed in Appendix C.

The only valid value of MODE for DBUPDATE calls is 1.

For users with access mode 2, DBUPDATE locks the data set DSET, physically reads the block containing the record to be updated, performs the update and writes the block back to disc and then unlocks the data set. DBUPDATE will return a CCL and a negative condition word if a user with access mode 2 without multiple RIN capability has some other resource locked.

5-23. DBPUT

DBPUT is employed to add new entries to a data base.

5-24. DECLARATION

```
PROCEDURE  DBPUT (BASE,DSET,MODE,
             STATUS,LIST,BUFFER);
ARRAY BASE,DSET,STATUS,LIST,BUFFER;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-25. OPERATION FOR MASTER DATA SETS

- The user is verified to have write access to the master data set referenced by DSET of the data base BASE. The data set is verified to be a MANUAL master having free space.
- The key item is verified as being referenced by LIST and its value in the BUFFER is checked for uniqueness.
- The new entry is added either as a primary entry or a secondary entry. In the latter case, synonym chain maintenance occurs which links it as the new last entry of the secondary chain. The fields of the new entry contain the data item values in the order defined in the schema, regardless of the order of their occurrence in LIST. Fields of unreferenced data items are filled with binary zeroes.
- Ten words, reflecting the new status of the master data set, are placed in the STATUS array as follows:

Words	Contents
1	A condition word of zero.
2	The word length of the logical entry in BUFFER.
3-4	The doubleword record address of the new entry.
5-6	The doubleword chain length of the synonym chain length of the synonym chain of the new entry.
7-8	The doubleword record address of the predecessor on the synonym chain.
9-10	A doubleword zero.

- A condition code of CCE is returned to the user.

5-26. OPERATION FOR DETAIL DATA SETS

- The user is verified to have write access to the detail data set referenced by DSET, of the data base BASE. The data set is verified to have free space.
- All search and sort items, if any, defined for the data set are verified as being referenced by LIST.
- Each related master data set, if any, is verified to contain a matching entry for the corresponding search item value. All MANUAL masters must have matching entries. If any AUTOMATIC master does not have a matching entry, it must have space to add one. This addition occurs automatically, and the current, internally maintained, synonym chain information for the AUTOMATIC master is zeroed.
- The new detail entry is added at an assigned record address. The fields of the new entry contain the data item values in the order defined in the schema regardless of the order of their occurrence in LIST. Fields of unreferenced data items are filled with binary zeroes.
- The new entry is linked into all corresponding chains. It is linked to the end of chains having no sort items; otherwise it is linked in its appropriate logical position within each chain as determined by the collating sequence of the sort items of the entries in each chain. This position is determined by a backward chain search starting at the last entry.
- Each chain head in the related master data sets is updated to reflect the addition of the entry to the corresponding chains of the detail data set.
- Ten words, reflecting the new internal status of the detail data set, are placed in the STATUS array as follows:

Words	Contents
1	A condition word of zero.
2	The word length of the logical entry in BUFFER.
3-4	The doubleword record number of the new entry.
5-6	The doubleword new chain length of the current chain of the new entry.
7-8	The doubleword record number of the predecessor on the current chain.
9-10	The doubleword record number of the successor on the current chain.

- A condition code of CCE is returned to the user.

Values of condition words corresponding to CCG and CCL condition codes are described in Appendix C.

The value of MODE must be 1.

The user must not have access mode 2 and the data set referenced by DSET must not be an AUTOMATIC master.

5-27. DBDELETE

DBDELETE is invoked to delete an entry of a data set.

5-28. DECLARATION

```
PROCEDURE  DBDELETE (BASE,DSET,MODE,
                     STATUS);
ARRAY BASE, DSET,STATUS;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-29. OPERATION FOR MASTER DATA SETS

- The user is verified to have write access to the master data set specified by DSET of the data base BASE.
- The entry, if any, at the current address of the referenced data set is deleted, providing that all chain heads, if any, indicate that the chains are empty.
- If the entry is a secondary entry, the media record is zeroed and marked empty. The required linkage maintenance is performed and the synonym chain count in the primary entry is decremented.
- If the entry is a primary entry with no synonyms, the media record is zeroed and marked empty.
- If the entry is a primary entry with synonyms, the first secondary is moved to the primary location, the record at the secondary address is zeroed and marked empty. The required linkage maintenance is performed and the synonym chain count in the new primary entry is decremented.
- Ten words, reflecting the new status of the master data set, are placed in the STATUS array as follows:

Words	Contents
1	A condition word of zero.
2	A data length of zero.
3-4	The unchanged current record address.
5-6	A zero chain count, unless the deleted entry was a primary with synonyms. In this case the chain count is one less than its previous value.
7-10	The unchanged predecessor and successor numbers unless the new chain count is not zero. In this case they reference respectively, the last and first secondaries, if any.

- A condition code of CCE is returned to the user.

5-30. OPERATION FOR DETAIL DATA SETS

- The user is verified to have write access to the detail data set specified by DSET of the data base BASE.
- The entry, if any, at the current address of the referenced data set is deleted by zeroing the media record, marking it empty, and adding it to the top of the "delete" chain for this data set.
- The required linkage maintenance, if any, is performed including modifications to the chain heads residing in the matching entries of related master data sets. If, as a consequence, the chain head in any AUTOMATIC master is deleted, the current, internally maintained, synonym chain information for the AUTOMATIC master is zeroed.
- Ten words, reflecting the new status of the detail data set, are placed in the STATUS array as follows:

Words	Contents
1	A condition word of zero.
2	A data length of zero.
3-10	Four doublewords constituting the current record number, chain count, and predecessor and successor record numbers unchanged since the preceding access to this data set.

- A condition code of CCE is returned to the user.

Values of condition words corresponding to CCG and CCL condition codes are described in Appendix C.

The value of MODE must be 1.

The user must not have access mode 2.

5-31. DBLOCK

DBLOCK provides the user with access mode 1 with the means of gaining temporary exclusive control of a data base.

5-32. DECLARATION

```
PROCEDURE  DBLOCK (BASE,DSET,MODE,
                   STATUS);
ARRAY BASE,DSET,STATUS;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-33. OPERATION

- The data set referenced by DSET, of the data base BASE, is verified to be accessible to the user.
- The root file is locked and the data base enabled for data access.
- A condition code of CCE and a condition word of zero is returned to the user.

Various condition words corresponding to CCG and CCL conditions are described in Appendix C.

MODE = 1, Unconditional Lock

In this mode, DBLOCK returns to the user only after exclusive control to the root file (and hence data base) has been acquired. This will occur only after the user currently controlling the data base, and any others queued up waiting their turn, release control by unlocking the data base.

MODE = 2, Conditional Lock

In this mode, DBLOCK returns to the user immediately whether he has acquired control of the data base or not. If he has acquired control, a CCE and condition word of 0 results. If not, a CCG and a condition word of 20 is returned, signifying that some other process has control of the data base.

A valid call by a user with access mode 2 or 3, or a redundant call by a user with access mode 1, is ignored.

5-34. DBUNLOCK

DBUNLOCK provides the user with access mode 1 with the means of relinquishing the temporary exclusive control of a data base acquired by a previous DBLOCK.

5-35. DECLARATION

```
PROCEDURE  DBUNLOCK (BASE,DSET,MODE,
                     STATUS);
ARRAY BASE,DSET,STATUS;
INTEGER MODE;
OPTION CHECK 2,EXTERNAL;
```

5-36. OPERATION

- The data set referenced by DSET, of the data base BASE, is verified to be accessible to the user.
- The root file is unlocked and the data base disabled for data access. Each data set is logically closed.
- A condition code of CCE and a condition word of zero is returned to the user.

Various condition words corresponding to CCG and CCL conditions are described in Appendix C.

The only valid value of MODE is 1.

A valid call by a user with access mode 2 or mode 3, or a redundant call by a user with access mode 1, is ignored.

CREATING AND MAINTAINING DATA BASES

SECTION VI

The Data Base Utilities are a set of utility programs designed to operate on an IMAGE/3000 data base. The five programs and their functions are:

DBUTIL
Allocates and initializes all data sets of a data base, or reinitializes all data sets of a data base, or purges the root file and all data sets of a data base.

DBSTORE
Store the root file and all data sets of a data base on a magnetic tape in a format compatible with tapes created by the MPE/3000 STORE or SYSDUMP commands.

DBRESTOR
Restores the root file and all data sets from a tape created by the DBSTORE program (or by the MPE/3000 STORE or SYSDUMP commands) to disc.

DBUNLOAD
Dumps all the data entries of each data set of a data base to a specially formatted magnetic tape. This logical dump of the data base is useful for changing the data base structure.

DBLOAD
Loads an already created data base with the contents of a DBUNLOAD tape.

6-1. MAINTENANCE WORD

The creator of a data base can authorize maintenance access for other users by specifying a **maintenance word** while running the DBUTIL program in CREATE mode. Subsequent utility users are defined access to the data base unless they specify the correct maintenance word. If the creator does not specify a maintenance word when he creates the data base, subsequent utility users (other than the creator) are totally denied access to the data base.

6-2. USING THE UTILITY PROGRAMS

The utility programs run in either Job or Session mode. The user must log on the same group in which the data base files are catalogued. The user then invokes a specific utility program by entering an MPE/3000 :RUN command (the exact format of which is described under the discussion for each utility program). In Job mode, the record immediately folliwng the :RUN command must be the **data base reference.** In Session mode, the user types the **data base reference** in response to the message:

WHICH DATA BASE?

Data base reference is defined as:

data base name [/maintenance word]

Data base name must match the name of an IMAGE/3000 root file which is catalogued in the user's log on group. The **maintenance word** is an ASCII string, one to eight characters long, with no commas or semicolons. A Session mode user may terminate execution by entering a carriage return (with or without leading blanks) instead of a **data base reference.**

6-3. DBUTIL

The utility program DBUTIL performs one of three possible functions, depending upon the program entry point chosen by the user.

6-4. FUNCTION

- If the user chooses the CREATE entry point, DBUTIL allocates disc space for each data set of the referenced data base and initializes this space to an empty (zeroed) condition. Each data set is saved as a catalogued MPE/3000 file. If the root file is named **XXXX**, then the first data set file is catalogued by the name **XXXX01**, the second data set file is catalogued by the name XXXX02, and so on. The user of DBUTIL (in CREATE mode) must be the creator of the data base root file.
- If the user chooses the ERASE entry point, DBUTIL reinitializes the data sets of the data base back to their empty condition. The data sets remain as catalogued MPE/3000 files. The user must be the data base creator or must provide the maintenance word.

- If the user chooses the PURGE entry point, DBUTIL 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, the user must be the creator or must provide the maintenance word.

DBUTIL is invoked through the MPE/3000 command:

:RUN DBUTIL.PUB.SYS, { CREATE / ERASE / PURGE }

Upon successful completion, DBUTIL sends one of the following messages to the user **listfile**, depending upon which entry point (CREATE, ERASE, OR PURGE) was entered in the :RUN command:

DATA BASE CREATED
DATA BASE ERASED
DATA BASE PURGED

DBUTIL also sends monitoring messages to the user **listfile** if an unexpected event occurs during DBUTIL execution. The messages are:

NO ROOT — DBUTIL successfully purged the data set files **XXXX01**, ...**XXXXn**, but was unable to locate the root file **XXXX**.

XXXXk 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 **XXXXk** where **k** > **n**.

XXXXk MISSING — DBUTIL successfully purged the root file and all existing data sets but data set **XXXXk** (**k** ⩽ **n**), mentioned in the message is unexpectedly missing.

Error conditions causing unsuccessful completion are described in **Appendix D, Utility Error Message.**

6-5. EXAMPLES

Example A. (Session Mode)

```
:RUN DBUTIL.PUB.SYS,CREATE
WHICH DATA BASE? SCHOOL/SUPER
DATA BASE CREATED
END OF PROGRAM
```

The first line of the above example invokes the DBUTIL program in CREATE mode. DBUTIL responds by typing the WHICH DATA BASE? message. The user then enters the name of the data base (SCHOOL) followed by a maintenance word (SUPER). If SCHOOL is a root file created by the Schema Processor for a data base consisting of three data sets, DBUTIL creates, initializes and saves files named SCHOOL01, SCHOOL02, and SCHOOL03. These, then, constitute the empty data base.

Example B. (Job Mode)

```
:RUN DBUTIL.PUB.SYS,ERASE
SCHOOL/SUPER
DATA BASE ERASED
END OF PROGRAM
```

The first line of the above example invokes the DBUTIL program in ERASE mode. The data base name and maintenance word follow the :RUN command in the next record. If SCHOOL is the name of a data base previously CREATEd as in Example A above, DBUTIL reinitializes files SCHOOL01, SCHOOL02, and SCHOOL03 to their original empty condition.

Example C. (Job Mode)

```
:RUN DBUTIL.PUB.SYS,PURGE
SCHOOL
DATA BASE PURGED
END OF PROGRAM
```

The first line of the above example invokes the DBUTIL program in PURGE mode. The name of the data base follows the :RUN command in the next record. If the user is the data base creator, and the root file SCHOOL exists, DBUTIL purges the root file and any data sets named SCHOOL01,. . .,SCHOOL99. If the root file does not exist, DBUTIL nevertheless purges any data sets named SCHOOL01,. . .,SCHOOL99.

6-6. DBSTORE

6-7. FUNCTION

The utility program DBSTORE gains exclusive access to the referenced data base and stores the data base root file and all data sets to a magnetic tape in a format compatible with MPE/3000 STORE and SYSDUMP tapes. DBSTORE differs from the MPE/3000 STORE and SYSDUMP programs in that it handles IMAGE/3000 data bases only.

A user may store any data base for which he is the creator or for which he supplies the correct maintenance word.

DBSTORE is invoked through the MPE/3000 command:

:RUN DBSTORE.PUB.SYS

Once invoked, DBSTORE attempts to dump the root file and all of the data sets of the data base. DBSTORE signals completion with the message:

DATA BASE STORED

If **DBSTORE** is unsuccessful, an error message is sent to the user **listfile**. Consult **Appendix D, Utility Error Messages.**

6-8. EXAMPLE (JOB MODE)

```
:RUN DBSTORE.PUB.SYS
SCHOOL/SUPER
DATA BASE STORED
END OF PROGRAM
```

The first line of the above example invokes the DBSTORE utility. If the root file named SCHOOL (with a maintenance word named SUPER) is present, DBSTORE stores the root file and existing data sets of the data base SCHOOL on magnetic tape. (The maintenance word is ignored if the user is the data base creator.)

6-9. DBRESTOR*

6-10. FUNCTION

The utility program DBRESTOR copies a data base from a tape created by the DBSTORE (or MPE/3000 STORE or SYSDUMP) program to disc. The files thus created are catalogued as MPE/3000 files.

DBRESTOR is invoked through the MPE/3000 command:

:RUN DBRESTOR.PUB.SYS

Once invoked, DBRESTOR attempts to copy the data base from tape to disc. If a catalogued disc file exists with the same name as the root or any data set of the data base, DBRESTOR does not copy the data base to disc and sends an error message to the user **listfile.** Consult **Appendix D, Utility Error Messages.**

If all files are successfully copied to disc and catalogued, DBRESTOR signals completion with the message:

DATA BASE RESTORED

6-11. EXAMPLE (SESSION MODE)

```
:RUN DBRESTOR.PUB.SYS
WHICH DATA BASE? SCHOOL/SUPER
DATA BASE RESTORED
END OF PROGRAM
```

The above example restores the data base SCHOOL. If the user was the creator of SCHOOL the maintenance word SUPER is ignored; otherwise, it is required. DBRESTOR copies the data base from tape to disc.

6-12. DBUNLOAD → STILL DUMP

6-13. FUNCTION

The utility program DBUNLOAD gains exclusive access to the referenced data base and copies data entries of each data set to a specially formatted, possibly multireel tape. (The root file is not copied to tape.) The information written to the tape consists of the data entry of each non-empty record. None of the pointer information associated with the data entry is tranferred.

DBUNLOAD operates in either chained or serial mode. In serial mode, DBUNLOAD copies the data entries serially in record number order. In chained mode, DBUNLOAD copies all of the detail entries with the same primary path search item value to continguous locations on tape. The ordering of the search item values corresponds to their physical ordering in the primary master data set of the detail data set being unloaded.

"Stand-alone" detail data sets, (i.e. those which are not tied to any master data sets through specified search item paths), are always unloaded serially.

DBUNLOAD is invoked through the MPE/3000 command:

:RUN DBUNLOAD.PUB.SYS **[,entry point]**

where

entry point
is either CHAINED or SERIAL. If **entry point** is omitted, DBUNLOAD operates in chained mode by default.

After each data set is copied to tape, DBUNLOAD sends the following message to the user **listfile**;

DATA SET **n**: **x** ENTRIES

where

n
is the data set number and

x
is the number of data set entries copied to tape.

If the number of entries copied to tape differs from the expected number of entries, DBUNLOAD sends the additional message

*NOTE: A program file name must be eight characters or less. The "E" has been dropped from the utility name for this reason.

*****NUMBER OF ENTRIES EXPECTED y

where

y
is the number of entries expected.

If an unrecoverable tape write error occurs, DBUNLOAD attempts to recover from the beginning of the current reel after sending the following message to the user **listfile**:

UNRECOVERABLE TAPE ERROR*
RESTARTING AT BEGINNING OF REEL

DBUNLOAD also sends the following message to the system operator to instruct him to mount a new reel for the write retry:

TAPE WRITE PROBLEMS TRY ANOTHER
REEL ON LOGICAL DEVICE **n**

where

n
is the logical device number of the tape drive.

If an excessive number of non-fatal tape write errors occur, DBUNLOAD again attempts to recover from the beginning of the reel after sending the following message to the user **listfile**:

EXCESSIVE TAPE ERROR RECOVERIES,
RESTARTING AT BEGINNING OF REEL

DBUNLOAD also sends a message to instruct the operator to mount a new reel for the write retry (same as unrecoverable tape error operator message).

Upon encountering an end of reel, DBUNLOAD sends the following message to the user **listfile**:

END OF REEL **a**, **b** TAPE ERRORS RECOVERED

where

a
is the number of the tape reel, and

b
is the number of tape write errors from which DBUNLOAD successfully recovered.

DBUNLOAD also instructs the operator to save the current reel and mount a new reel with the following two messages:

SAVE TAPE ON LOGICAL DEVICE **n** AS **XXXX y**

MOUNT NEXT REEL ON LOGICAL DEVICE **n**

where

n
is the logical device number of the tape drive,

XXXX
is the name of the data base, and

y
is the reel number.

After the data sets have been successfully copied to tape, DBUNLOAD sends the following message to the user **listfile**:

DATA BASE UNLOADED

6-14. EXAMPLE (JOB MODE)

```
:RUN DBUNLOAD.PUB.SYS
SCHOOL
DATA SET 1:   9 ENTRIES
DATA SET 2:  13 ENTRIES
DATA SET 3:  28 ENTRIES
END OF REEL 1,  0 TAPE ERRORS RECOVERED
DATA BASE UNLOADED
END OF PROGRAM
```

If the data base named SCHOOL is present, and the user is the creator of the data base, DBUNLOAD unloads the data base in CHAINED (which is default) mode.

6-15. DBLOAD

6-16. FUNCTION

The utility program DBLOAD acquires exclusive access to a data base and loads logical records from magnetic tape into data sets of the data base. The magnetic tape used must be in the exact format as produced by the DBUNLOAD program, and the data base name on the tape must be exactly the same as the data base name on disc.

Logical records on the tape, constituting the data set entries, are loaded into a corresponding data set on the disc. The entries of the first data set on the tape are loaded (using the DBPUT library procedure) into the first data set on disc, etc.

If the data set logical record is larger than the tape logical record, the data set record is padded on the right with binary zeros. If the data set logical record is smaller than the tape logical record, the tape record is truncated on the right and the truncated data is lost.

To reload the identical data into a data base that was previously dumped to tape, the user must execute the DBUTIL program, ERASE option, between execution of the DBUNLOAD and DBLOAD programs. Executing DBUTIL in this way reinitializes the data sets to an empty state while keeping the root file and data sets as catalogued MPE/3000 files on the disc.

DBLOAD is invoked through the MPE/3000 command:

```
:RUN DBLOAD.PUB.SYS
```

DBLOAD attempts to load the logical tape records into the data base. After each data set is loaded, DBLOAD sends the following message to the user **listfile**:

```
DATA SET n: x ENTRIES
```

where

n
is the number of the data set, and

x
is the number of entries transferred from the tape. (Automatic master entries are not loaded; instead they are created when related detail sets are loaded.) This number may not represent the total number of records residing in the data set if entries existed in the set prior to DBLOAD execution.

If the data base to be loaded exists on more than one reel of tape, DBLOAD sends the following message to the user **listfile** at the end of each reel:

```
END OF REEL n, x TAPE ERRORS RECOVERED
```

where

n
is the reel number and

x
is the number of tape read errors from which DBLOAD recovered.

DBLOAD also sends the following message to the operator to instruct him to mount the next reel:

```
MOUNT DBLOAD TAPE XXXX y
ON LOGICAL DEVICE n
```

where

n
is the logical device number of the tape drive,

XXXX
is the data base name, and

y
is the reel number.

If the operator mounts the wrong tape on the tape drive, DBLOAD informs the operator with the following message:

```
WRONG TAPE MOUNTED ON
LOGICAL DEVICE n
```

where

n
is the logical device number of the tape drive.

After the last reel of tape is successfully loaded into the data base, DBLOAD sends the additional message to the user **listfile**.

```
DATA BASE LOADED
```

Any other error conditions accompanying unsuccessful completion are described in **Appendix D, Utility Error Messages**.

6-17. EXAMPLE (SESSION MODE)

```
:RUN DBLOAD.PUB.SYS
WHICH DATA BASE? SCHOOL/SUPER
DATA SET 1:   9 ENTRIES
DATA SET 2:   0 ENTRIES
DATA SET 3:  28 ENTRIES
END OF REEL 1,  0 TAPE ERRORS RECOVERED
DATA BASE LOADED
END OF PROGRAM
```

The entries on the DBUNLOAD tape externally called SCHOOL 1 are loaded into data base SCHOOL, if it is present and has a maintenance word SUPER (or if the user is the creator). No entries are explicity put into the second data set, since it is an automatic master.

SAMPLE DATA BASE APPLICATION

APPENDIX A

Figure A-1 depicts a sample data base which might be used by a department store to keep track of credit card purchases and inventory. The data base consists of two detail data sets and four masters.

A master data set is used to contain information about each charge account customer. A single entry, whose key item is charge account number, represents a customer. A second master data set has one entry for each of the store's products, containing the product's stock number (the key item) and its description.

A detail data set is used to contain information about credit card purchases. Each entry represents a purchase by a particular customer of a certain quantity of an individual product. This data set has search items designated which establish paths between it and both of the previously described masters. Also, two items within each entry contain date information, and are designated as search items, doubly linking this data set to an automatic master whose key field is date.

A master data set is employed to contain information on each company which supplies products to the store. In this data set, the supplier's name is the key field. A detail data set contains inventory information (one entry per product), and search items link it to the supplier master data set and the date master data set, as well as to the product master data set.

Figure A-2 contains the schema for the department store data base, as listed by the Schema Processor. After successfully creating the root file, as shown, the data base creator must allocate and initialize the data sets by using the DBUTIL program as explained in Section VI.

The remaining figures in this appendix illustrate access to the data base from user-written programs containing calls to the IMAGE/3000 library procedures. Each of the three programs illustrated runs in Session mode and opens and accesses the department store data base.

The programs, illustrate calls to the Library procedures from host languages. Exceptional conditions returned by the procedures which represent expected errors (such as bad input values typed by a user) cause the programs to print appropriate messages and continue operation. Unexpected errors or conditions cause the programs to print the contents of the STATUS area and then to terminate.

Figure A-3 contains an SPL/3000 program called SUPPLMOD which opens the data base in DBOPEN 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 A-4.

Figure A-5 is an SPL/3000 program called SHOWSALE, which displays credit card purchase transactions from the detail data set containing these entries. SHOWSALE opens the data base in DBOPEN mode 2 thereby avoiding the necessity of locking and unlocking the data base. Figure A-6 shows th output from SHOWSALE.

Figure A-7 is COBOL/3000 program called RECEIVE which updates on-hand quantities and adjusts unit costs in the data set containing inventory information. Like SHOWSALE, RECEIVE opens the data base in DBOPEN mode 2. Sample output from RECEIVE is contained in Figure A-8.

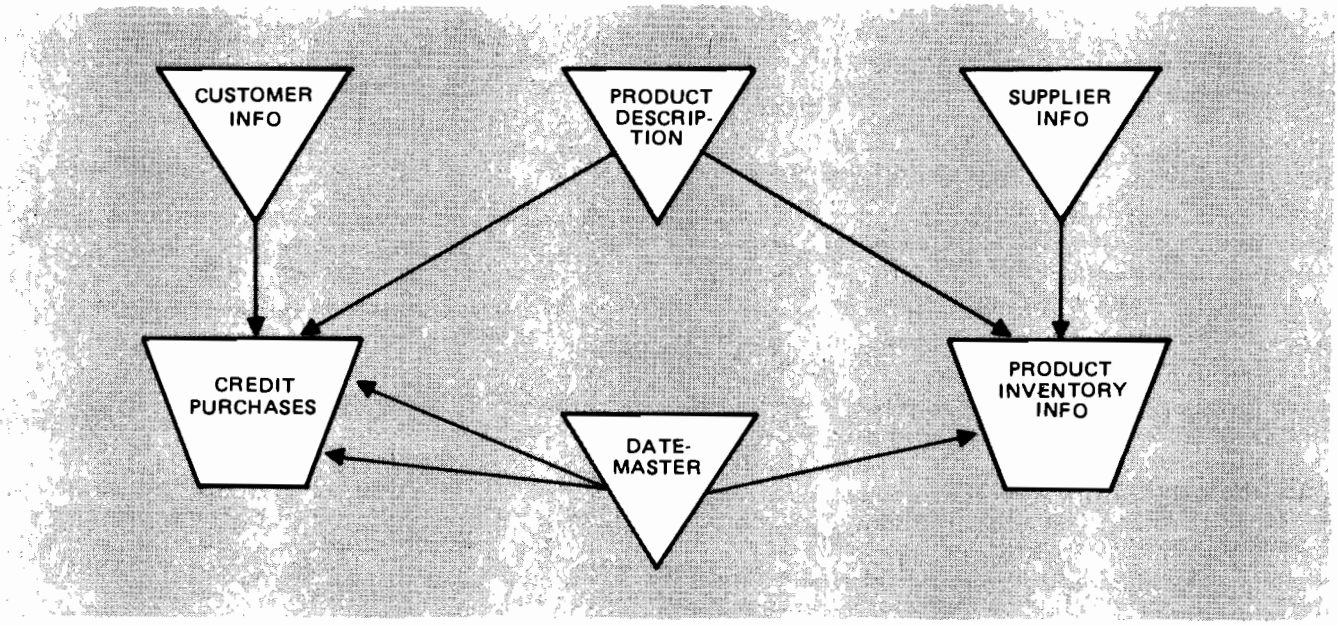

Figure A-1. Department Store Data Base

```
PAGE 1      HEWLETT-PACKARD 32215X.00    IMAGE/3000  THU, JUN 13, 1974,

$CONTROL BLOCKMAX=512,LINES=56

BEGIN DATA BASE STORE;

     LEVELS:
       2 CLERK;
       4 BUYER;
       8 SUPER;
      10 JR-EXEC;
      13 MANAGER;
      15 PRESDENT;

     ITEMS:                 << IN ALPHABETICAL ORDER FOR CONVENIENCE >>
      ACCOUNT,          K2   (2,12);  << CUSTOMER ACCOUNT NUMBER >>
      CITY,             X12  (2,8);   << CITY >>
      CREDIT-RATING,    K    (10,10); << CUSTOMER CREDIT RATING >>
      DATE,             X6 ;          << DATE (YYMMDD) >>
      DELIV-DATE,       X6 ;          << DELIVERY DATE (YYMMDD) >>
      DESCRIPTION,      X20  (2,4);   << PRODUCT DESCRIPTION >>
      FIRST-NAME,       X10  (2,8);   << CUSTOMER GIVEN NAME >>
      INITIAL,          X2   (2,8);   << CUSTOMER MIDDLE INITIAL >>
      LAST-NAME,        X16  (2,8);   << CUSTOMER SURNAME >>
      LASTSHIPDATE,     X6   (4,4);   << DATE LAST RECEIVED (YYMMDD) >>
      ONHANDQTY,        J2   (4,4);   << TOTAL PRODUCT INVENTORY >>
      PRICE,            J2   (0,8);   << SELLING PRICE (PENNIES) >>
      PURCH-DATE,       X6 ;          << PURCHASE DATE (YYMMDD) >>
      QUANTITY,         I    (0,8);   << SALES PURCHASE QUANTITY >>
      STATE,            X2   (2,8);   << STATE -- 2 LETTER ABBREVIATION >>
      STOCK#,           X8   (2,12);  << PRODUCT STOCK NUMBER >>
      STREET-ADDRESS,   X26  (2,8);   << NUMBER AND STREET >>
      SUPPLIER,         X14;          << SUPPLYING COMPANY NAME >>
      TAX,              J2   (0,8);   << SALES TAX (PENNIES) >>
      TOTAL,            J2   (0,8);   << TOTAL AMOUNT OF SALE (PENNIES) >>
      UNIT-COST,        J2   (4,4);   << UNIT COST OF PRODUCT (PENNIES) >>
      ZIP,              X6   (2,8);   << ZIP CODE >>

     SETS:

      NAME:     CUSTOMER,MANUAL(2,10);   << CUSTOMER MASTER INFO >>
      ENTRY:    ACCOUNT(1),
                LAST-NAME,
                FIRST-NAME,
                INITIAL,
                STREET-ADDRESS,
                CITY,
                STATE,
                ZIP,
                CREDIT-RATING;
      CAPACITY: 2003;

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

Figure A-2. Department Store Data Base Schema

```
PAGE 2       STORE


      NAME:     PRODUCT,MANUAL(2,10);     << PRODUCT INDEX >>
      ENTRY:    STOCK#(2),
                DESCRIPTION;
      CAPACITY: 5003;

      NAME:     SALES,DETAIL(2,8);        << CREDIT PURCHASE INFO >>
      ENTRY:    ACCOUNT(CUSTOMER(PURCH-DATE)),
                STOCK#(PRODUCT),
                QUANTITY,
                PRICE,
                TAX,
                TOTAL,
                PURCH-DATE(DATE-MASTER),
                DELIV-DATE(DATE-MASTER(ACCOUNT));
      CAPACITY: 12000;

      NAME:     SUP-MASTER,MANUAL(4,13); << SUPPLIER MASTER INFO >>
      ENTRY:    SUPPLIER(1),
                STREET-ADDRESS,
                CITY,
                STATE,
                ZIP;
      CAPACITY: 601;

      NAME:     INVENTORY,DETAIL(4,10);  << PRODUCT SUPPLY INFO >>
      ENTRY:    STOCK#(PRODUCT),
                ONHANDQTY,
                SUPPLIER(!SUP-MASTER),         << PRIMARY PATH >>
                UNIT-COST,
                LASTSHIPDATE(DATE-MASTER);
      CAPACITY: 5000;

END.


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

CUSTOMER          M   2  10   9    1   40    50    2003      10   501   808
DATE-MASTER       A   0  15   1    3   3     23    211       22   508   44
PRODUCT           M   2  10   2    2   14    29    5003      13   378   1158
SALES             D   2  8    8    4   19    35    12012     14   491   3436
SUP-MASTER        M   4  13   5    1   30    40    601       12   481   208
INVENTORY         D   4  10   5    3   18    30    5015      17   512   1184
                                  TOTAL DISC SECTORS INCLUDING ROOT: 6846


NUMBER OF ERROR MESSAGES: 0
HIGHEST LEVEL WORD: 15      ITEM NAME COUNT: 22      DATA SET COUNT: 6
ROOT LENGTH: 716     BUFFER LENGTH: 512     TRAILER LENGTH: 256

ROOT FILE STORE  CREATED.
```

Figure A-2 (continued)

```
$CONTROL USLINIT,LINES=56,MAIN=SUPPLMOD

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 ACCESS TO IT.                       >>
<<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>

INTEGER    MODE1      := 1,      << MODES FOR      >>
           MODE7      := 7,      <<  USE IN        >>
           MODE201    := 201,    <<   IMAGE CALLS >>
           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 LEVELWD(0:3)   := "MANAGER;";       << QUALIFIER -- LEVEL WORD >>
ARRAY DSETNAME(0:5)  := "SUP-MASTER;";    << QUALIFIER -- DATA SET NAME >>
ARRAY STATUS(0:9);                        << STATUS AREA >>
ARRAY SUPBUF(0:29);                       << 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,DBDELETE,DBUNLOCK,DBCLOSE;
INTRINSIC READ,PRINT,ASCII,QUIT;

     << BEGINNING OF MAIN PROGRAM >>

          DBOPEN(SBASE,LEVELWD,MODE1,STATUS);
               << OPEN STORE DATA BASE IN MODE 1. >>
          IF <> THEN GO TO DBERROR;

          DBINFO(SBASE,DSETNAME,MODE201,STATUS,DSET);  << GET NUMBER  >>
          IF <> THEN GO TO DBERROR;                    << OF SUP-MASTER,>>
          DSET := \DSET\;              << MAKE SURE DSET# IF POSITIVE>>
```

Figure A-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,(29);         <<   BUFFER,        >>

          PRINT(PROMPT,5,%320);            << REQUEST AND READ  >>
          READ(SUPBUF,-14);                <<   SUPPLIER NAME,  >>
          IF FUNCTION = "D" THEN GO TO LOCKIT;  << DELETE; GO DO IT, >>
          PRINT(PROMPT(5),4,%320);         << REQUEST AND READ  >>
          READ(SUPBUF(7),-26);             <<   STREET ADDRESS, >>
          PRINT(PROMPT(9),3,%320);
          READ(SUPBUF(20),-12);            <<   CITY,           >>
          PRINT(PROMPT(12),-7,%320);
          READ(SUPBUF(26),-2);             <<   STATE,          >>
          PRINT(PROMPT(16),-5,%320);
          READ(SUPBUF(27),-5);             <<   AND ZIP CODE,   >>

LOCKIT:   DBLOCK(SBASE,DSET,MODE1,STATUS);    << LOCK DATA BASE, >>
          IF <> THEN GO TO DBERROR;
          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 DBERROR;

          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 DBERROR;
          GO TO UNLOCKIT;

NEWSUP:   DBPUT(SBASE,DSET,MODE1,STATUS,FULLREC,SUPBUF);
          IF <> THEN
            IF STATUS = 16 THEN PRINT(SETFULL,18,0)    << NO ROOM >>
```

Figure A-3 (continued)

```
              ELSE IF STATUS = 43 THEN PRINT(DUPE,17,0)  << DUPLICATE >>
                ELSE GO TO DBERROR;

UNLOCKIT:  DBUNLOCK(SBASE,DSET,MODE1,STATUS);
           IF = THEN GO TO ASK;

DBERROR:   << COME HERE ON UNEXPECTED OR IRRECOVERABLE ERROR >>
           <<   RETURNED BY IMAGE PROCEDURE.                  >>
           OUTBUF := "  ";                       << BLANK OUTPUT >>
           MOVE OUTBUF(1) := OUTBUF,(39);        <<   BUFFER.    >>
           FOR I := 0 UNTIL 9 DO ASCII(STATUS(I),8,BOUTBUF(I*8));
                << TRANSLATE STATUS WORDS INTO OUTBUF FOR PRINTING. >>
           PRINT(OUTBUF,20,0);              << OUTPUT STATUS AREA >>
           PRINT(OUTBUF(20),20,0);          <<   ON TWO LINES.    >>
           QUIT(1);                    << IRRECOVERABLE: GET OUT. >>

OUT:       DBCLOSE(SBASE,DSET,MODE1,STATUS);
           IF <> THEN GO TO DBERROR;
           END.
```

Figure A-3 (continued)

```
: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 A-4. Sample SUPPLMOD Execution

```
$CONTROL USLINIT,LINES=57,MAIN=SHOWSALE

BEGIN

<<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
<< THIS PROGRAM OPENS THE "STORE" DATA BASE IN                            >>
<<   MODE 2   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.                             >>
<<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>

INTEGER   MODE,
          MODE1     := 1,     << MODES       >>
          MODE2     := 2,     <<  FOR        >>
          MODE4     := 4,     <<   USE       >>
          MODE5     := 5,     <<    IN       >>
          MODE7     := 7,     <<     IMAGE   >>
          MODE201   := 201,   <<      CALLS  >>
          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 LEVELWD(0:2)  := "CLERK;";        << QUALIFIER -- LEVEL WORD >>
ARRAY SALENAME(0:2) := "SALES;";        << QUALIFIER -- DATA SET NAME >>
ARRAY PRODNAME(0:3) := "PRODUCT;";      << QUALIFIER -- DATA SET NAME >>
ARRAY STATUS(0:9);                      << STATUS AREA >>
```

Figure A-5. Purchase Transaction Display Program

```
ARRAY SALESLIST(0:25) := "ACCOUNT,QUANTITY,STOCK#,TOTAL,",
                         "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;
INTRINSIC READ,PRINT,ASCII,QUIT,DASCII,DBINARY;

     << BEGINNING OF MAIN PROGRAM >>

          DBOPEN(SBASE,LEVELWD,MODE2,STATUS);
               << OPEN STORE DATA BASE IN MODE 2. >>
          IF <> THEN GO TO DBERROR;

          DBINFO(SBASE,SALENAME,MODE201,STATUS,SALES);
               << FOR EFFICIENCY, GET NUMBER OF SALES DATA SET. >>
               IF <> THEN GO TO DBERROR;
          SALES := \SALES\;          << MAKE SURE DSET# IS POSITIVE. >>
          DARG := 0D;
          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 DBERROR;

          DBINFO(SBASE,PRODNAME,MODE201,STATUS,PRODUCT);
          IF <> THEN GO TO DBERROR;
          PRODUCT := \PRODUCT\;
          DBGET(SBASE,PRODUCT,MODE4,STATUS,PRODLIST,OUTBUF,DARG);
          IF STATUS <> 12 THEN GO TO DBERROR;
               << ALSO SET UP FOR DBGETS FROM PRODUCT DATA SET. >>

NEXT;     GRANDTOTAL := 0D;
          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>>
```

Figure A-5 (continued)

```
            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.                                  >>

            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>>
              ELSE GO TO DBERROR;
            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.                           >>
```

Figure A-5 (continued)

```
          MODE := MODE2;                    << PREPARE FOR SERIAL DBGETS. >>
          DARG := 1D;
          DBGET(SBASE,SALES,MODE4,STATUS,SAMELIST,SALESBUF,DARG);
               << "REWIND" SALES DATA SET BY READING ENTRY #1. >>
          IF = THEN GO TO PREPLINE;         << ENTRY #1 IS PRESENT >>
          IF STATUS <> 17 THEN GO TO DBERROR;

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.                >>
          IF <> THEN
            IF STATUS = 11 OR STATUS = 15 THEN GO TO WRAPUP << NO MORE>>
            ELSE GO TO DBERROR;

PREPLINE: << 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# >>
          DBGET(SBASE,PRODUCT,MODE7,STATUS,SAMELIST,OUTBUF(13),
                    SALESBUF(3));  << GET DESCRIPTION FROM PRODUCT >>
          IF <> THEN GO TO DBERROR;                <<  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. >>

DBERROR:  << COME HERE ON UNEXPECTED OR IRRECOVERABLE ERROR >>
          <<   RETURNED BY IMAGE PROCEDURE.                   >>
          OUTBUF := "  ";                      << BLANK OUTPUT >>
          MOVE OUTBUF(1) := OUTBUF,(39);       <<   BUFFER.    >>
          FOR I := 0 UNTIL 9 DO ASCII(STATUS(I),8,BOUTBUF(I*8));
               << TRANSLATE STATUS WORDS INTO OUTBUF FOR PRINTING. >>
          PRINT(OUTBUF,20,0);               << OUTPUT STATUS AREA >>
          PRINT(OUTBUF(20),20,0);           <<   ON TWO LINES.    >>
          QUIT(1);                    << IRRECOVERABLE: GET OUT. >>

OUT:      DBCLOSE(SBASE,SALES,MODE1,STATUS);
          IF <> THEN GO TO DBERROR;
          END.
```

Figure A-5 (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 A-6. Sample SHOWSALE Execution

```
$CONTROL  LIST,MAP,LINES=57

*     *     *     *     *     *     *     *     *     *     *
*                    -   C O M M E N T   -
*       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 LEVEL
*   WORD USED WAS "BUYER" SINCE THE TWO FIELDS BEING CHANGED HAVE
*   4 AS A WRITE LEVEL.  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 CONCURRENTLY.  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 INPUTTING TRANSACTIONS.  ONE CONSEQUENCE OF THIS IS
*   THAT THE USER MUST HIT CARRIAGE RETURN TWICE IF HIS RESPONSE
*   IS LESS THAN 8 CHARACTERS LONG.
*     *     *     *     *     *     *     *     *     *     *

 IDENTIFICATION DIVISION.
 PROGRAM-ID.  RECEIVE.
 DATE-COMPILED.
        THU, JUN 13, 1974,  9:08 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 9(9) COMP.
 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   LEVEL          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.
   10    STAT4-5     PIC 9(9) COMP.
   10    STAT6-7     PIC 9(9) COMP.
```

Figure A-7. Inventory Update Program

```
  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.
PROCEDURE DIVISION.
FIRST-PARAGRAPH-NAME.
    CALL "DBOPEN" USING BASE-NAME, LEVEL, 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 = "DEBUG" CALL "DEBUG"   GO TO ASK-FOR-IP.
    IF STOCK-NO = "SAYONARA" GO TO FINISH.
    PERFORM FIND-STOCK-RECORD.
    IF CONDTN-WORD = 17 OR = 15
      DISPLAY "NO SUCH STOCK NUMBER"
      GO TO ASK-FOR-IP.
    MOVE  SPACES TO STOCK-NO.
    DISPLAY "NOW ENTER QUANTITY RECIEVED - ".
    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.
    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 STOP RUN.
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  STOP RUN.
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.
```


Figure A-7 (continued)

```
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   STOP RUN.
DISPLAY-STATUS.
     MOVE CONDTN-WORD TO STATUS-EDIT
     DISPLAY   "CONDITION WORD - " STATUS-EDIT.
     MOVE STAT1 TO STATUS-EDIT DISPLAY "STATUS 1 - "  STATUS-EDIT.
     MOVE STAT2-3 TO STATUS-EDIT DISPLAY "STATUS 2-3 - "
     STATUS-EDIT.
     MOVE STAT4-5 TO STATUS-EDIT DISPLAY "STATUS 4-5 - "
     STATUS-EDIT.
     MOVE STAT6-7 TO STATUS-EDIT DISPLAY "STATUS 6-7 - "
     STATUS-EDIT.
     MOVE STAT8-9 TO STATUS-EDIT DISPLAY "STATUS 8-9 - "
     STATUS-EDIT.
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 A-7 (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 A-8. Sample RECEIVE Execution

SCHEMA PROCESSOR ERROR MESSAGES

APPENDIX B

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 B-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/3000 file information display on the $STDLIST file. Consult **HP 3000 Multiprogramming Executive Operating System**, 03000-90005 for the meaning of MPE/3000 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 IV), will cause the Schema Processor to terminate execution.

Table B-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 B-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 IV), 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 B-1. File Errors

MESSAGE	MEANING
UNABLE TO USE **file name**	The specified file cannot be FOPENed, or its characteristics make it unsuitable for its intended use.
READ ERROR ON **file name**	An FREAD error has occurred on the specified file.
WRITE ERROR ON **file name**	An FWRITE error has occurred on the specified file.
UNABLE TO WRITE LABEL OF **file name**	An FWRITELABEL error has occurred on the specified file.
UNEXPECTED END-OF-FILE ON **file name**	A call to FREAD or FWRITE on the specified file has yielded an unexpected end of file condition.
UNABLE TO CLOSE **file name**	An FCLOSE error has occurred on the specified file.

Table B-2. Schema Processor Command Errors

MESSAGE	MEANING
ILLEGAL COMMAND	The schema processor does not recognize the command.
IMPROPER COMMAND PARAMETER	One of the parameters in a command is not valid.
COMMAND CONTINUATION NOT FOUND	If a schema processor command is continued to the next record, the last non-blank character of the preceding line must be an ampersand (&), and the continuation record must start with a dollar sign ($).
MISSING QUOTATION MARK	The character string specified in a PAGE or TITLE command must be bracketed by quotation marks ('').
COUNT HAS BAD FORMAT	The numbers appearing in the ERRORS, LINES, BLOCKMAX parameters of the CONTROL command are not properly formatted integer values.
SPECIFIED TITLE TOO LONG	The character string appearing in the TITLE command is too long.

Table B-3. Schema Syntax Errors

MESSAGE	MEANING
AUTOMATIC MASTER MUST HAVE SEARCH ITEM ONLY	AUTOMATIC master data sets must contain entries with only one data item, and that data item must be the search item.
BAD CAPACITY OR TERMINATOR	Either the number in the CAPACITY: statement is not an integer between 1 and 2^{23}-1 or a semicolon is missing.
BAD PATH COUNT OR TERMINATOR	The path count in the master data set definition is not an integer 1 to 16 (for an automatic) or an integer from 0 to 16 (for a manual). Note this message may occur because the path count is not followed by a ")".
BAD CHARACTER IN LEVEL NUMBER	A level number in the level part is not an integer from 1 to 15.
BAD DATA BASE NAME OR TERMINATOR	The data base name in the BEGIN DATA BASE statement is not a valid data base name of from one to six alphanumeric characters beginning with an alphabetic, or it is not followed by a semicolon.
BAD DATA SET TYPE	The data set type designator is not AUTOMATIC (or A), MANUAL (or M), or DETAIL (or D).
BAD PATH CONTROL PART DELIMITER	The data item defined as a sort item in a detail data set is not properly delimited with parentheses.

Table B-3. Schema Syntax Errors (Continued)

MESSAGE	MEANING
BAD PATH SPECIFICATION DELIMITER	The name of a master data set following a 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 LEVEL OR TERMINATOR	The read level number defined for either a data set or data item is not an integer from 0 to 15 or is not separated from the write level number by a comma.
BAD SET NAME OR TERMINATOR	A data set name does not conform to the rules for data set names given in Section IV, or is not terminated by the correct character for the context in which it appears.
BAD SUBITEM COUNT OR TERMINATOR	The subitem count for a data item defined in the schema item part is not an integer from 1 to 255.
BAD SUBITEM LENGTH OR TERMINATOR	The subitem length for a data item defined in the schema item part is not an integer from 1 to 255.
BAD TERMINATOR - ';' or ',' EXPECTED	The items within an entry definition must be separated from each other by commas and terminated by a semicolon.
BAD TERMINATOR - ';' EXPECTED	A level word or capacity was not followed by a semicolon.
BAD TYPE DESIGNATOR	A data item defined in the schema item part is not defined as type I, J, K, R, U, X, Z, or P.
BAD WRITE LEVEL OR TERMINATOR	The write level number indicated for either a data set or a data item is not an integer greater than or equal to the associated read level less than 16 or is not terminated by a right parenthesis.
'CAPACITY:' EXPECTED	A CAPACITY statement must follow the entry definition in the definition of a data set in the set part of a schema.
DATA BASE HAS NO DATA SETS	No data sets were defined in the set part of the schema. A data base must contain at least one data set.
DETAIL WRITE LEVEL GREATER THAN MASTER WRITE LEVEL	The write level of a detail data set must be less than or equal to the write level of any related master data set.
DUPLICATE ITEM SPECIFIED	The same data item name is used more than once in the entry definition of a data set.
DUPLICATE ITEM NAME	A data item name appears more than once in the item part of the schema.
DUPLICATE SET NAME	The same name has been used to define more than one data set in the schema set part.
'ENTRY:' EXPECTED	Each data set defined in the schema set part must contain an ENTRY: statement followed by data item names of data items in the entry.

Table B-3. Schema Syntax Errors (Continued)

MESSAGE	MEANING
ENTRY TOO BIG	The number and size of data items defined for an entry of a data set yields an entry which is too large for the maximum block size (as either specified by the $CONTROL BLOCKMAX= command or by default).
ENTRY TOO SMALL	A detail data set that is not linked to any master data sets must have a data entry length equal to or greater than two words. This length is determined by adding the size in words of each data item defined in the data entry.
ILLEGAL LEVEL NUMBER	The level number defined in the schema level part is not in ascending order or is not an integer between 1 and 15, inclusive.
ILLEGAL ITEM NAME OR TERMINATOR	A data item name does not conform to the naming standards described in Section IV, or, if in the item part, is not followed by a comma.
ITEM LENGTH NOT INTEGRAL WORDS	A data item, simple or compound, must occupy an integral number of words.
ITEM TOO LONG	A single data item cannot exceed 2047 words in length.
'LEVELS:' NOT FOUND	The 'LEVELS:' statement must immediately follow the BEGIN DATA BASE statement in the schema. If it does not, DBSCHEMA terminates execution.
LEVEL WORD TOO LONG	Any level word defined in the data schema cannot exceed eight characters.
MASTER DATA SET LACKS EXPECTED DETAILS	A master data set was defined with a non-zero path count, but the number of detail search items which back-referenced the master is less than the value of path count.
MASTER DATA SET LACKS SEARCH ITEM	A master data set was defined without defining one of the data items in the set as a key (search) item.
MASTER SEARCH ITEM PRIVACY LOOPHOLE	A detail data set's search item can be read with an access level less than that necessary to read the corresponding master data set's search item. This is disallowed.
MORE THAN ONE KEY ITEM	A master data set cannot be defined with more than one search item.
MORE THAN ONE PRIMARY MASTER	The user has defined more than one primary path for a detail data set.
'NAME:' OR 'END.' EXPECTED	The Schema Processor expected at this point to encounter the beginning of another data set definition or the end of the schema.
REFERENCED SET NOT A MASTER	The data set referenced by a detail data set search item is another detail data set rather than a master.

Table B-3. Schema Syntax Errors (Continued)

MESSAGE	MEANING
SCHEMA PROCESSOR LACKS NEEDED TABLE SPACE	The 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 the schema correctly, the operating system must be configured with a larger maximum stack size.
SEARCH ITEM NOT SIMPLE	All data items defined in the data schema as search items must be simple items.
SEARCH ITEMS NOT OF SAME LENGTH	A master search item must be the same length as any related detail data set search item.
SEARCH ITEMS NOT OF SAME TYPE	A master search item must be the 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 the master data set's path count.
SORT ITEM NOT IN DATA SET	A detail data sets entry definition does not include an item which is specified as the sort item for another item in the entry.
SORT ITEM OF BAD TYPE	The data item defined as a sort item must be of type U, K, or X.
SORT ITEM SAME AS SEARCH ITEM	The same item cannot be both search and sort item for the same path.
TOO MANY DATA ITEMS	The item part of a schema may contain no more than 255 data item names.
TOO MANY DATA SETS	A data base can contain no more than 99 data sets.
TOO MANY ERRORS	The specified or default maximum number of errors has been exceeded. Processing terminates.
TOO MANY ITEMS SPECIFIED	A data set entry can contain no more than 127 data items.
TOO MANY PATHS IN DATA SET	A detail data set entry can contain no more than 16 search items.
UNDEFINED ITEM REFERENCED	A data item appearing in a data set definition was not previously defined in the item part of the data schema.
UNDEFINED SET REFERENCED	A master data set referenced by a detail search item has not been previously defined in the set part of the data schema.

LIBRARY PROCEDURE ERROR MESSAGES

APPENDIX C

The success of each call to an IMAGE/3000 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 C-1 describes the negative condition words resulting from file system and memory management failures, while Table C-2 describes the negative condition words resulting from calling errors.

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 C-3 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.

C-1. ABORT CONDITIONS

In general, four types of error conditions can cause IMAGE/3000 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/3000 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/3000 run-time abort message as described in the **HP 3000 Multiprogramming Executive Operating System,** 03000-90005. In cases 2, 3, and 4, IMAGE/3000 prints additional information on the user's standard list device about the error prior to printing the standard MPE/3000 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 c-4 describes additional lines of information which may appear prior to the standard MPE/3000 abort message.

Some of the abort conditions are due to an error in one of the MPE/3000 file intrinsics FOPEN, FREADLABEL, FREADDIR, FWRITELABEL, FWRITEDIR, FLOCK, 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 C-4 which refer to an IMAGE/3000 data file is followed by an MPE/3000 file information display which lists all of the characteristics of the MPE/3000 data set or root file where the error occurred, along with an MPE/3000 error number. For more information about file error codes and the file information display, consult **HP 3000 Multiprogramming Executive Operating System,** 03000-90005.

Table C-1. File System and Memory Management Errors

CONDITION WORD	DESCRIPTION
-1	FOPEN intrinsic failure. For DBOPEN, this error may indicate that the data base could not be opened. Possible reasons for this are: • The user failed to terminate the data base name string with a semicolon or a blank • The data base does not exist or is secured against access by its group or account security • The data base is already opened exclusive or in a mode different from the requested mode • A file system error has occurred For DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE, this error may occur if: • The user has too many files open external to the data base • A data set doesn't exist or is somehow secured against access • Some other file system error has occurred
-2	FCLOSE failure. This error is exceptional (should never happen) and is returned only by DBOPEN or DBCLOSE. It indicates a hardware or system software failure.
-3	FREADDIR failure. This error is exceptional (as with -2 above) and is returned by DBOPEN, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE.
-4	FREADLABEL failure. This error is exceptional (as with -2 above) and is returned by DBOPEN, DBINFO, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE.
-7	FLOCK failure. Returned by DBLOCK (or DBUPDATE if the user opened the data base in Mode 2) if the user does not have multiple RIN capability and he already has been assigned a RIN or, rarely, if all of the system's global RIN's have been assigned.
-9	GETDSEG failure. This exceptional failure is returned by DBOPEN when it cannot obtain an extra data segment for use as the Data Base Control Block. this occurs if the required virtual memory space is unavailable or if the required DST entry is unavailable.

Note: For condition words -1 through -4 and -7, the second word of the user's status area is the number of the data set for which the file error occurred (zero indicates the root file) and the third word is the MPE/3000 failure code returned by the FCHECK intrinsic.

For condition code -9, the second word of the user's status area is the size (in words) of the data segment which memory management was unable to supply, and the third word is the MPE/3000 failure code returned by the GETDSEG intrinsic.

Consult **HP 3000 Multiprogramming Executive Operating System, 03000-90005** for the meaning of the MPE/3000 failure codes.

Table C-2. Calling Errors

CONDITION WORD	DESCRIPTION
-11	Bad BASE parameter. For DBOPEN, the first two characters of BASE are not blank, or the data base name contains special characters other than period. For all other library procedures, either the first two characters of BASE do not contain the value assigned by DBOPEN, or exceptionally, the parameters passed to the procedure are incorrect in type, sequence or quantity.
-12	Global data base name. For DBOPEN, the BASE parameter contains more than one period. Users are not permitted to access data bases across accounts. This error may be caused by failure to terminate the data base name with a blank or semicolon. Data base not enabled. For DBFIND, DBGET, DBUPDATE, DBPUT, and DBDELETE, the data base has been opened in DBOPEN Mode 1 but has not been locked (using DBLOCK) since being opened or since the last successful DBUNLOCK. Bad access mode. For DBPUT and DBDELETE, the data base has been opened in DBOPEN Mode 2. These procedures may not be used with access mode 2.
-21	Bad level word. For DBOPEN, the access level granted the user is too low to permit access to any data in the data base. This is usually due to an incorrect or null level word. Bad data set reference. For DBINFO (modes 104, 201, 202, 301, and 302), DBCLOSE, DBFIND, DBGET, DBUPDATE, DBPUT, DBDELETE, DBLOCK, and DBUNLOCK, when a data set reference is either: • Numeric but out of range • An erroneous data set name • A reference to a data set which is inaccessable to the user. An erroneous data set name may arise when a terminating semicolon or blank is omitted. Bad data item reference. For DBINFO (in modes 101, 102, and 204), the data item reference is either: • Numeric but out of range • An erroneous data item name • A reference to a data item which is inaccessable to the user. An erroneous data item name may arise when a terminating semicolon or blank is omitted.
-23	Data set not writable. For DBPUT and DBDELETE, the data base has been opened in DBOPEN Mode 1 or Mode 3, and the user has read but not write access to the referenced data set.
-24	Data set is an automatic master. For DBPUT, the referenced data set is an automatic master.

Table C-2. Calling Errors (Continued)

CONDITION WORD	DESCRIPTION
-31	Bad MODE. This error occurs in all library procedures when the MODE parameter is invalid. For DBGET, MODE is 7 or 8 and the referenced data set is a detail, or MODE is 5 or 6 and the referenced data set is a stand alone detail (i.e., a detail data set without search items.)
-51	Bad LIST length. For DBGET, DBUPDATE, and DBPUT, the list is too long. This may occur if the list is not terminated with a semicolon or blank. It may also occur for otherwise legitimate, lists which are too long for the trailer area; it will never occur for numeric lists.
-52	Bad LIST or bad ITEM. For DBGET, DBUPDATE, and DBPUT, the LIST parameter contains a data item reference which either: • is invalid (out of range) • references an inaccessable data item • duplicates another reference in the list For DBFIND, the ITEM parameter contains a data item reference which either: • is invalid (out of range) • references an inaccessable data item • is not a search item for the referenced data set
-53	Missing search or sort item. For DBPUT, a search or sort item of the referenced data set is not included in the LIST parameter.
-91	Bad root modification level. For DBOPEN, the software version of the DBOPEN procedure is incompatable with the version of the Schema Processor which created the root file.
-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).

Table C-3. Exceptional Conditions

CONDITION WORD	DESCRIPTION
10	Beginning of file. DBGET has encountered the beginning of file during a backward serial read. (There are no entries before the one previously accessed.)
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 the data set.
14	Beginning of chain. DBGET has encountered the beginning of chain during a backward chained read.
15	End of chain. DBGET has encountered the end of chain during a forward chained read.
16	Data set full. DBPUT has discovered that the data set is full.
17	No entry. Either DBGET has encountered an empty record in response to a directed read or DBFIND is unable to locate the master entry for the specified search item value.
20	Data base locked. DBLOCK (in conditional mode) has discovered that the data base is locked by another process.
41	Critical item. DBUPDATE has been asked to change the value of a search or sort item.
42	Read only item. DBUPDATE has been asked to change the value of a data item for which the user does not have write access.
43	Duplicate search item value. DBPUT has been asked to insert a data entry into a master data set with a search item value which already exists in the data set.
44	Chain head. DBDELETE has been asked to delete a master data set entry which still has one or more non-empty detail chains.

Table C-3. Exceptional Conditions (Continued)

CONDITION WORD	DESCRIPTION
50	Buffer too small. The user's buffer (identified by BUFFER) is too small for the amount of information that DBGET or DBINFO wishes to return.
1**xx**	Missing chain head. The user has attempted to add a detail data entry with a search item value that does not match any existing search item value in the corresponding manual master data set. The digits **xx** identify the offending path number.
2**xx**	Full chain. The user has attempted to add a detail data entry to a chain which already contains the maximum allowable (65535) entries. The digits **xx** identify the offending path number.
3**xx**	Full master. The user has attempted to add a detail data entry with a search item value that does not match any existing search item value in the corresponding automatic master data set and a new master entry cannot be created because the automatic master data set is full. The digits **xx** identify the offending path number.

Note: For condition words in the 1xx, 2xx, and 3xx series, path number (as specified by xx) refers to paths by the order in which their search items occur in the entry part of the schema.

Table C-4. Abort Messages

MESSAGE	DESCRIPTION
BAD ADDRESS FOR PARAMETER #**n**	The address referenced by one of the parameters is not within the user's stack area in memory (roughly between the DL and Q registers). **n** is the positional number of the parameter in the procedure's calling sequence. The first parameter is number 1, the second is number 2, etc.
UNABLE TO OPEN **data**set	The procedure was unable to open a data base file.
critical label read error on **data set**	The procedure was unable to read the label of a data base file.
CRITICAL READ ERROR ON **data set**	The procedure encountered an MPE/3000 file read error while reading a data base file.
LABEL WRITE ERROR ON **data set**	The procedure was unable to complete the writing of the user label of a data base file.
WRITE ERROR ON **data set**	The procedure encountered an MPE/3000 file write error while writing into a data base file.
UNABLE TO LOCK **data set**	The procedure was unable to lock a data base file.
UNABLE TO CLOSE **data set**	The procedure was unable to close a data base file.
LOST FREE SPACE IN **data set**	An internal software inconsistancy has causes unused record locations in the data set to become lost or unavailable. IMAGE/3000 out a file information display for the data set file.
BUFFER SUPPLY CRISIS	An internal software inconsistancy has caused IMAGE/3000 to improperly manage its buffer space.

UTILITY ERROR MESSAGES

APPENDIX D

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. Errors generating these messages can be corrected without terminating the run if the user is in Session mode. After printing the error message the utility reprompts the user with the message: "WHICH DATA BASE?", allowing the user to re-enter the data base reference. If the user wishes to terminate the utility program at this point he may merely type a carriage return, with or without leading blanks. If the user is in Job mode, conditional errors always cause program termination. Conditional error messages and their meanings ae described in Table D-1.

Unconditional errors occur in utility programs after successful execution has already begun. These errors always cause program termination. The accompanying messages and their meanings are described in Table D-2.

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 **HP 3000 Multiprogramming Executive Operating System** (03000-90005). Those initiated by the library procedures called by the utilities are explained in **APPENDIX C, LIBRARY PROCEDURE ERRORS.**

Table D-1. Utility Program Conditional Error Messages

MESSAGE	MEANING
BAD DATA BASE REFERENCE	The **data base reference** following the utility program :RUN command contains a syntax error.
BAD MAINTENANCE WORD	The user invoking the utility is not the creator of the referenced data base and has supplied an incorrect maintenance word.
DATA BASE ALREADY CREATED	The user has invoked the DBUTIL utility in CREATE mode and has specified the name of a data base which already exists.
DATA BASE IN USE	The utility programs cannot gain exclusive access to the referenced data base because of other current user(s).
DATA BASE REQUIRES CREATION	The data base creator must run the DBUTIL program in CREATE mode prior to invoking DBUNLOAD, DBLOAD, or DBUTIL in ERASE mode.
DUPLICATE FILE NAME	A required file name is already assigned to some other file. For example, if data base STORES requires six data sets (STORES01 — STORES06), then a previously defined file STORES03 would trigger this message.
EXCEEDS ACCOUNT DISC SPACE	The amount of disc space required by the data base plus that already assigned to other files of this account exceeds the amount of disc space available to the account.
EXCEEDS GROUP DISC SPACE	The amount of disc space required by the data base plus that already assigned to other files in this group exceeds the amount of disc space available to the group.
INSUFFICIENT DISC SPACE	The amount of disc space required for the data base is not available from the system.

Table D-1. Utility Program Conditional Error Messages (continued)

MESSAGE	MEANING
INSUFFICIENT VIRTUAL MEMORY	The amount of virtual memory available for program execution is insufficient.
MAINTENANCE WORD REQUIRED	The user invoking the utility is not the creator of the referenced data base and has failed to supply a maintenance word.
NO SUCH DATA BASE	The specified data base does not exist in the user's log on group.
NOT ALLOWED; MUST BE CREATOR	The user invoking the utility is not the creator of the data base and the data base has no maintenance word.
OUTMODED ROOT	The root file of the specified data base corresponds to a different version of the IMAGE/3000 software and is not compatible with the utility program currently executing.

Table D-2. Utility Program Unconditional Error Messages

MESSAGE	MEANING
AUTOMATIC MASTER IS FULL ON PATH NUMBER **n**	DBLOAD is unable to load a detail entry because the automatic master data set associated with path **n** of the detail data set is full.
CHAIN IS FULL ON PATH NUMBER **n**	DBLOAD is unable to load a detail entry because the chain count for path number **n** of the detail data set exceeds 2^{16}-1.
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/3000 library procedure. Parameters p_1,p_2,p_3 are usually the first three words of the status area returned by a library procedure. The values of the parameters depends upon the procedure and the specific error condition. Consult **Appendix C, LIBRARY PROCEDURE ERRORS** for the meanings of the parameter values. p_4 is the PCODE of the library procedure which encountered the error. The PCODE is a number uniquely assigned to the procedure to identify it. p_5 is an octal PB-relative return address within the code segment of the utility which initiated the error message. (p_1 through p_4 are decimal while p_5 is octal.) Consult Section V for description of the library procedures.
DATA SET FULL	The data set currently being loaded is full.
NO MANUAL ENTRY FOR DETAIL ON PATH NUMBER **n**.	DBLOAD is attempting to load a detail data set entry for which there is no matching manual master data set entry. **n** is the number of the detail data set path referencing the manual master in question.
UNABLE TO CONTINUE	The DBUTIL program (operating in PURGE mode) cannot continue execution due to an exceptional error in the file system. This information is followed by a file information display as defined in the **HP 3000 Multiprogramming Executive Operating System, 03000-90005.**

RESTRUCTURING DATA BASES

APPENDIX E

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 data base. The general sequence of operations which accomplishes this is as follows:

1. Run DBUNLOAD on the old data base, backing up all entries to tape.
2. Purge the old data base using DBUTIL, PURGE.
3. Redefine the data base (of the same name) and create a new root file using DBSCHEMA.
4. Create and initialize the data sets of the new data base by using DBUTIL, CREATE.
5. Run DBLOAD on the old data base using the tape created in step 1 to put the old data into the new base.

The data base design changes for which the above procedure will function correctly (yield the desired results) are limited. An understanding of the functioning of the DBUNLOAD and DBLOAD utilities is necessary in order to deduce the set of valid changes.

E-1. DBUNLOAD

DBUNLOAD writes each entry of each data set of a specified data base to magnetic tape. The entries of each data set are located in contiguous blocks on the tape, and the data sets are unloaded to tape in the order that they were defined in the original schema. No data set names are recorded on tape; entries are merely associated with the **number** of the data set from which they were read.

DBUNLOAD calls the DBGET procedure to read each entry from each set of the data base. The LIST parameter passed to DBGET is "@;", meaning that the full logical record is read and written to tape. DBUNLOAD always handles full entries, without regard to item positions or lengths. (Values for data items appear in each entry in the same order as the items were mentioned in the data set definition.)

DBUNLOAD never writes any data base structure information on tape.

E-2. DBLOAD

DBLOAD requires that its target data base has the same name, group, and account, and the same number of data sets as the data base on tape (produced by DBUNLOAD). DBLOAD reads each entry from the tape and puts it into the same numbered data set from which it was read by the DBUNLOAD program. If a data set in the receiving data base is an automatic master, no entries are directly put into it by DBLOAD, even through there may be entries on the tape associated with the data set's number. Automatic master entries will be created as needed in the normal fashion when entries are subsequently put into related detail data sets.

DBLOAD calls the DBPUT procedure to put the entries read from tape into the appropriate data sets of the data base. In every case, the DBPUT DSET parameter is a data set number (as specified on the tape), and the LIST parameter is "@;". Prior to calling DBPUT, DBLOAD moves each entry from the tape into a buffer (left justified) which is padded out to the maximum entry length with binary zeroes. The DBPUT procedure always uses as many words from the designated buffer as are required to supply concatenated values for the items specified by the LIST parameter. In the case of the special "full record" list (@;), the entry length used by DBPUT is the length of the full entry of the data set in the target data base. In the case of DBLOAD, this length may be less than, equal to, or greater than the length of the entries on the tape.

Changes made to a data base definition after unloading the data base and prior to recreating and reloading it must be made with the preceding information in mind. The new schema must, of course, meet all of the syntactical and semantical constraints of the schema processor. Assuming successful creation of the new root file and empty data base, the results of reloading the data fall into three general categories:

1. DBLOAD may fail. For example, a data set's capacity may have been reduced in the new data base to a number less than the number of that data set's entries on the tape.
2. DBLOAD may succeed, but resulting data item values are not all valid.

   For example, the new schema may contain items which are interchanged in a data set's entry definition. New items may have been added at the beginning or in the interior of a data entry definition. In these cases, "old" items in the new data base will not have the same values they did in the original data base.

3. DBLOAD may succeed, producing a correctly transformed data base. For example, a data set's capacity may have been increased, or a level word changed. The resulting data base contains completely valid data and all structural relationships remain the same.

Schema changes which 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 in combination) which are acceptable to the schema processor will always result in a successfully transformed data base:

1. Adding, changing, or deleting level words
2. Changing a data item or data set name and all references to it
3. Changing data item or data set read and write levels
4. Adding new data item definitions
5. Removing or changing definitions of unreferenced data items
6. Increasing data set capacities
7. Adding, deleting, or changing sort item designators
8. Changing primary paths
9. Adding new data items to the original end of a data entry definition
10. Removing data items from the original end of a data entry definition
11. 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. Specifically, All entries from old data set **n** are put into new data set **n** (except that no entries are directly put into automatic masters), and that entries are truncated or padded with zeroes as necessary to fit the new data set's entry length. DBUNLOAD and DBLOAD always handle full entires, without regard to item positions or lengths.

DATA BASE BACKUP AND RECOVERY

APPENDIX F

IMAGE/3000 software is designed to maintain and insure the integrity of IMAGE data bases. There is the possibility, however, of losing data or data base structure 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.

In general, data base backup and recovery cannot be accomplished independent of system backup and recovery. Each approach described below requires the absolute cooperation of all operations personnel.

One approach is simply to depend on a daily system dump and system reload, if needed, by operations personnel. This solution is a viable one if:

1. The user is able to recover from a midday system failure by rerunning all jobs which modified the data base.

2. The user is **immediately** informed of all system reloads so that he will not run additional jobs prior to bringing the reloaded data base up-to-date.

3. The user has access to the system dump tapes, so that he may restore the data base if needed after a system crash which is not followed by a system reload.

A second approach is for the user to maintain his data base independently. After each use, he must copy the data base to tape using DBSTORE and then purge it using DBUTIL. Before each use, he must restore the data base from tape, using DBRESTOR. This solution is particularly suitable when:

1. The previous method is untenable.

2. The data base does not need to be continuously on line.

3. The number of uses of the data base per day is small and the combined time of storing, purging and restoring is short relative to use time.

4. There are no other data bases or files which must be kept in synchronization with the user's data base.

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 and, if needed, to restore them collectively from tape with the RESTORE command.

It is worth noting that these three options are mutually exclusive and each merely restores the data base to its state at backup time. No automatic way is provided to cover the changes from backup time to crash time.

In practice, the latter 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. Logging of transactions by the 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, an installation must design backup and recovery procedures which meet its particular needs.

F-1. SALVAGING DATA

If a data base structure is damaged, and no backup tapes 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 the 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. If all necessary manual master data entries (if any) are written to tape, reloading the data base using the DBLOAD program and this tape (after erasing the data base using DBUTIL) will result in a structurally intact approximation of the original data base.

INDEX