HP 3000 Computer Systems

IMAGE
Data Base Management System

Reference Manual

HEWLETT PACKARD

19447 PRUNERIDGE AVE.,CUPERTINO,CALIFORNIA 95014

Part No. 32215-90003
Product No. 32215B

Printed in U.S.A 9/79

NOTICE

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

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

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

DATA BASE STRUCTURE AND PROTECTION

SECTION II

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

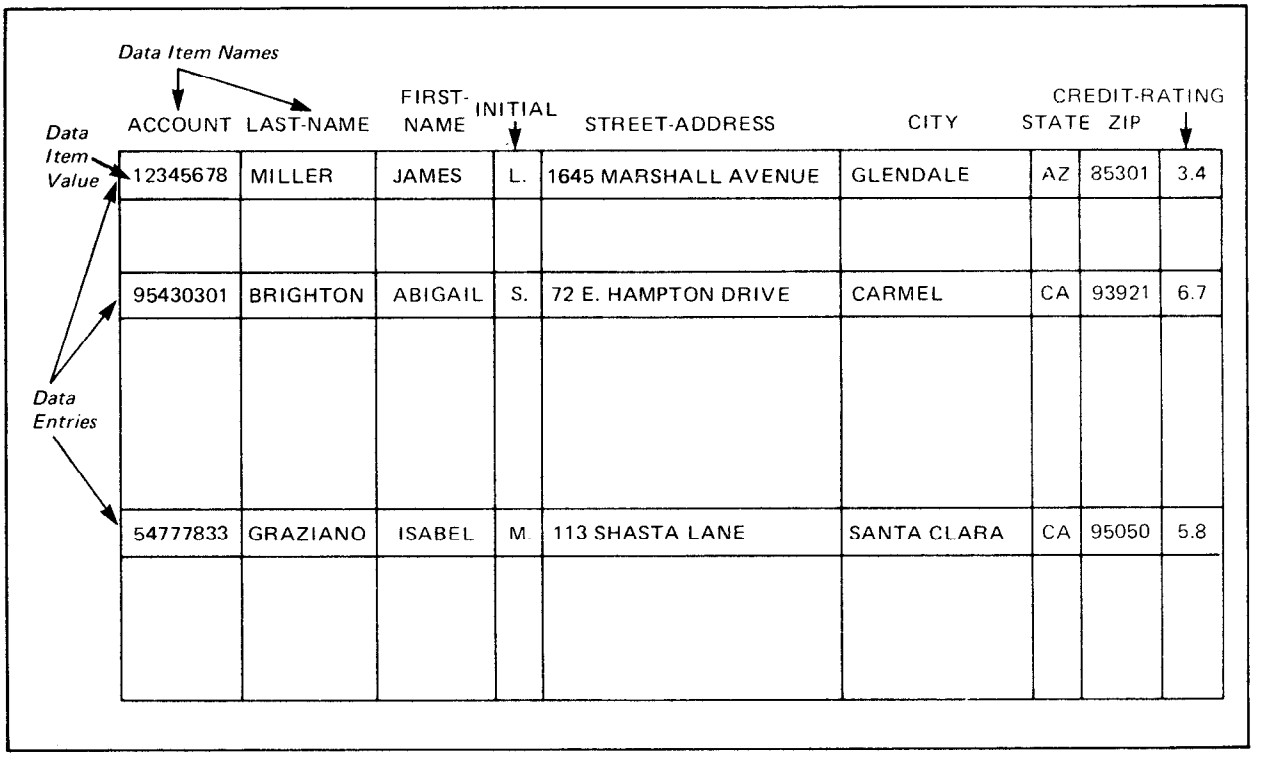
DATA ELEMENTS

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

DATA ITEMS

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

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



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

Figure 2-1. CUSTOMER Data Set Sample

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

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

DATA ENTRIES

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

DATA SETS

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

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

DATA SET TYPES AND RELATIONS

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

MASTER DATA SETS

Master data sets are characterized in the following ways:

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

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

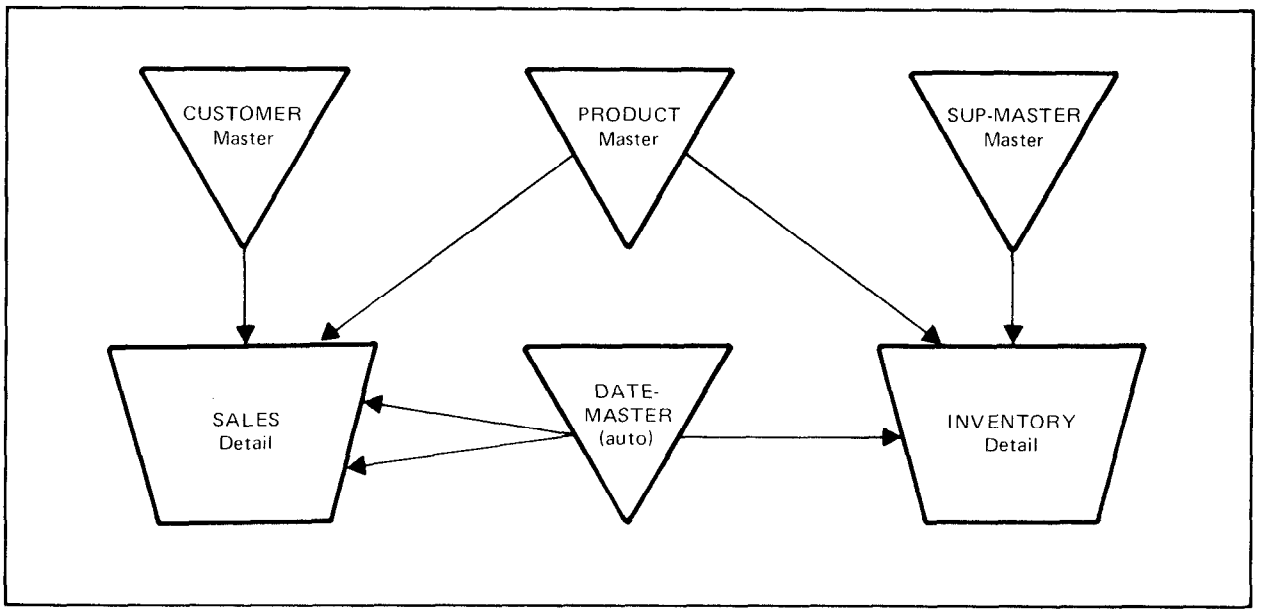


Figure 2-2. Master and Detail Data Set Relations

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

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

DETAIL DATA SETS

Detail data sets are characterized in the following ways:

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

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

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

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

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

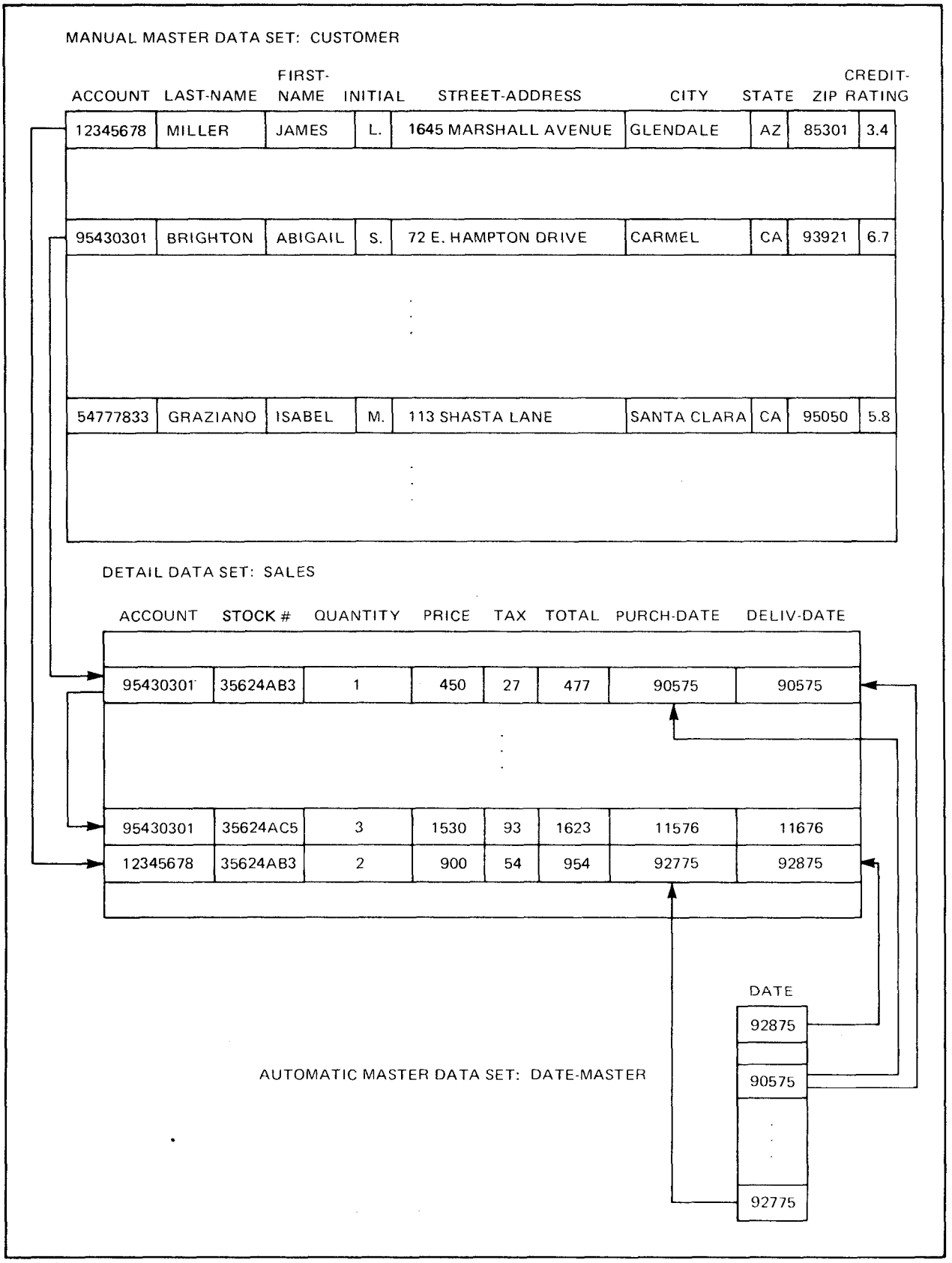


Figure 2-3. Master and Detail Data Sets Example

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

PATHS

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

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

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

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

AUTOMATIC AND MANUAL MASTERS

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

MANUAL	AUTOMATIC
May be stand-alone. Need not be related to any detail data set.	Must be related to one or more detail data sets.
May contain data items in addition to the search item.	Must contain only one data item, the search item.
You must explicitly add or delete all entries. A related detail data entry cannot be added until a master entry with matching search item value has been added. When the last detail entry related to a master entry is deleted, the master entry still remains in the data set. Before a master entry can be deleted, all related detail entries must be deleted.	IMAGE automatically adds or deletes entries when needed based on the addition or deletion of related detail data set entries. When a detail entry is added with a search item value different from all current search item values, a master entry with matching search item value is automatically added. Deletions of detail entries trigger an automatic deletion of the matching master entry if it is determined that all related data chains are empty.

MANUAL	AUTOMATIC
The search item values of existing master entries serve as a table of legitimate search item values for all related detail data sets. Thus, a non stand-alone manual master can be used to prevent the entry of invalid data in the related detail data sets.	

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

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

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

MANUAL VS. AUTOMATIC DATA SETS

Data base designers may use:

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

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

PRIMARY PATHS

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

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

SORT ITEMS

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

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

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

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

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

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

THE STORE DATA BASE

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

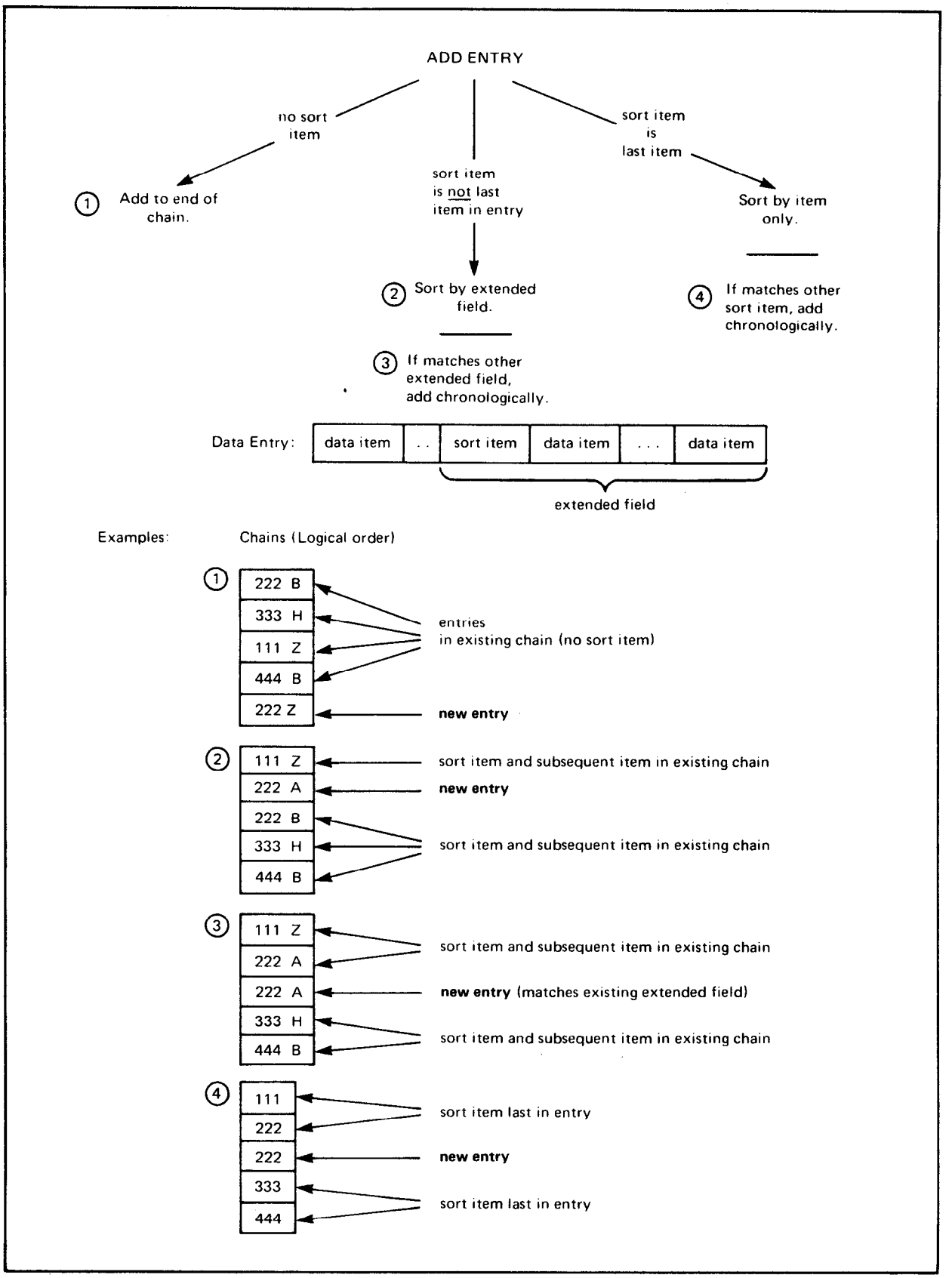


Figure 2-4. Adding an Entry to a Sorted Chain

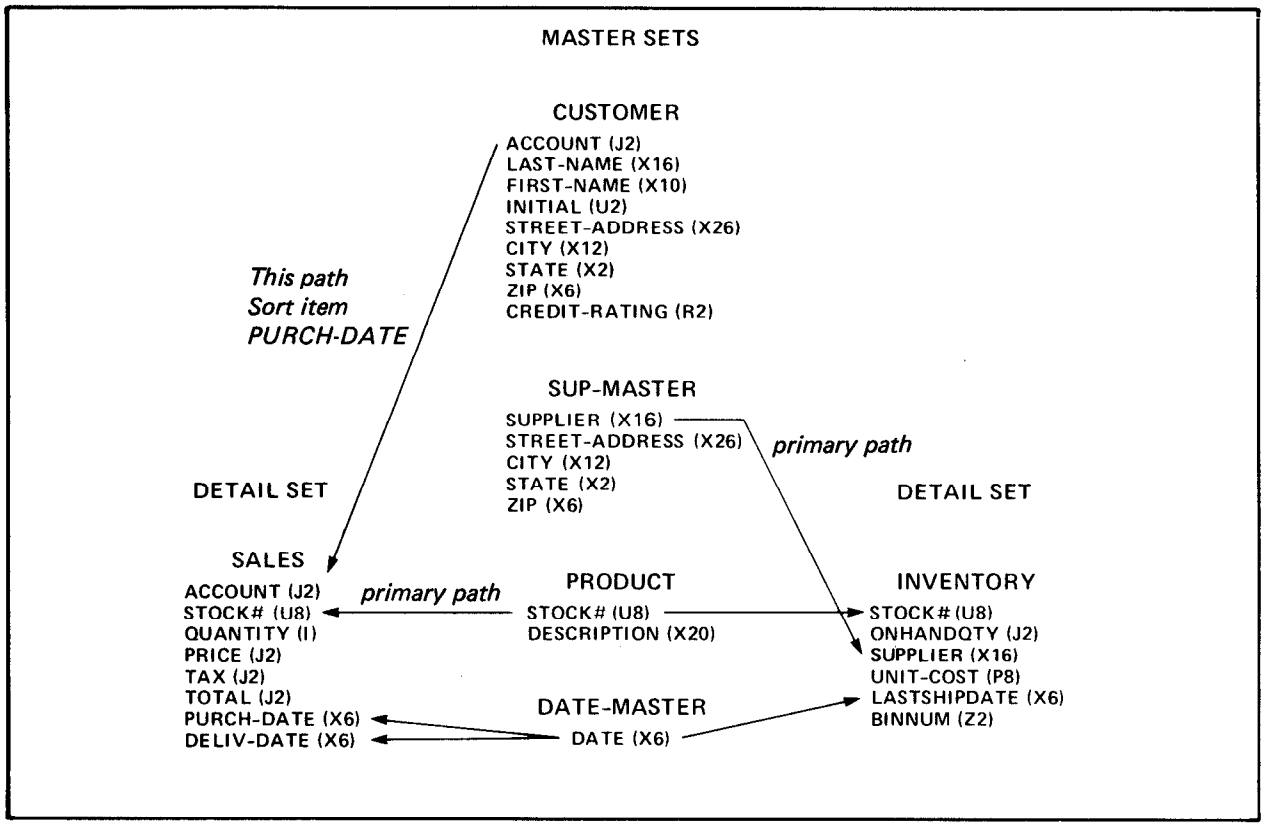


Figure 2-5. STORE Data Sets and Paths

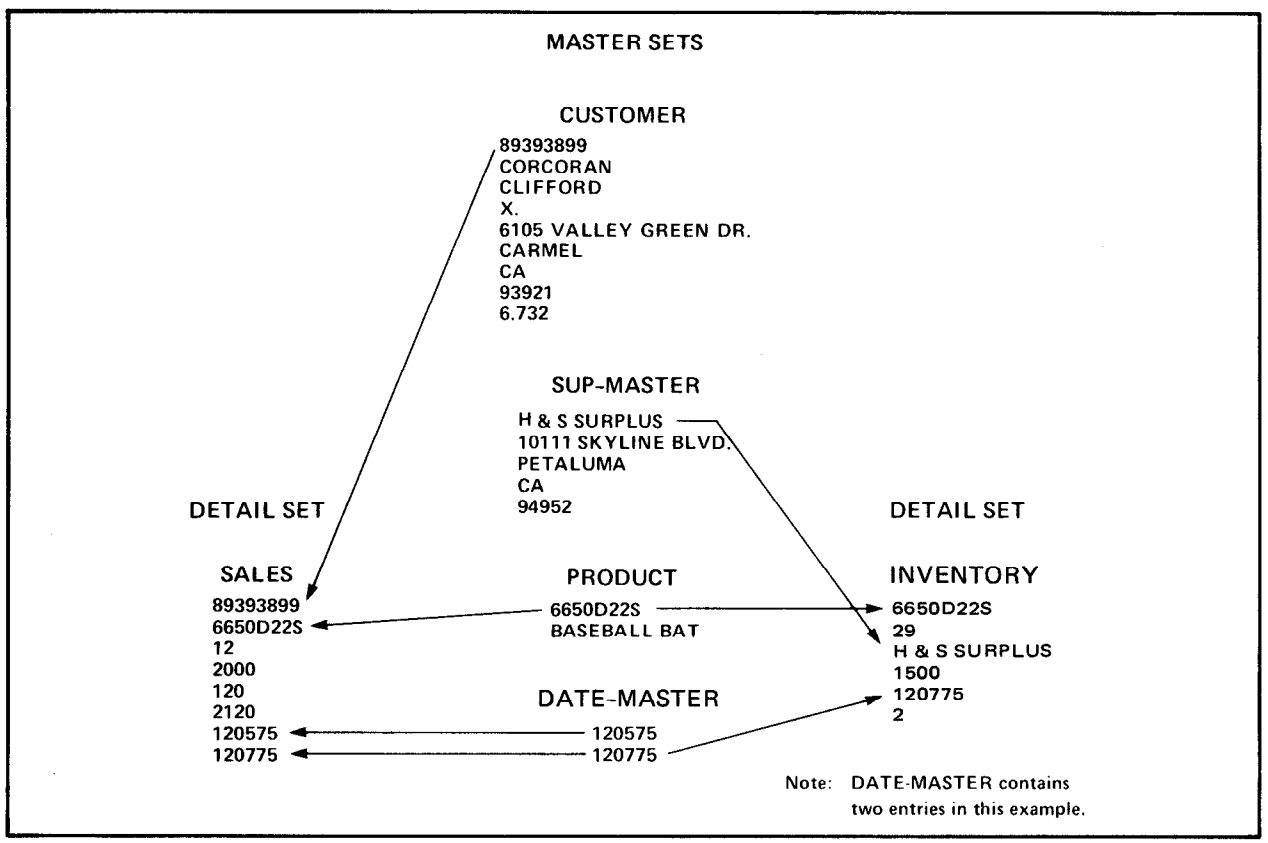


Figure 2-6. Sample Entries for STORE Data Sets

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

DATA BASE FILES

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

ROOT FILE

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

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

DATA FILES

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

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

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

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

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

PROTECTION OF THE DATA BASE

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

PRIVILEGED FILE PROTECTION

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

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

ACCOUNT AND GROUP PROTECTION

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

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

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

USER CLASSES AND PASSWORDS

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

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

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

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

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

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

Table 2-1. Sample Read/Write Class Lists

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

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

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

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

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

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

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

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

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

ACCESS MODES AND DATA SET WRITE LISTS

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

GRANTING A USER CLASS ACCESS TO A DATA ELEMENT

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

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

Table 2-2. Granting Capability to User Class 11

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

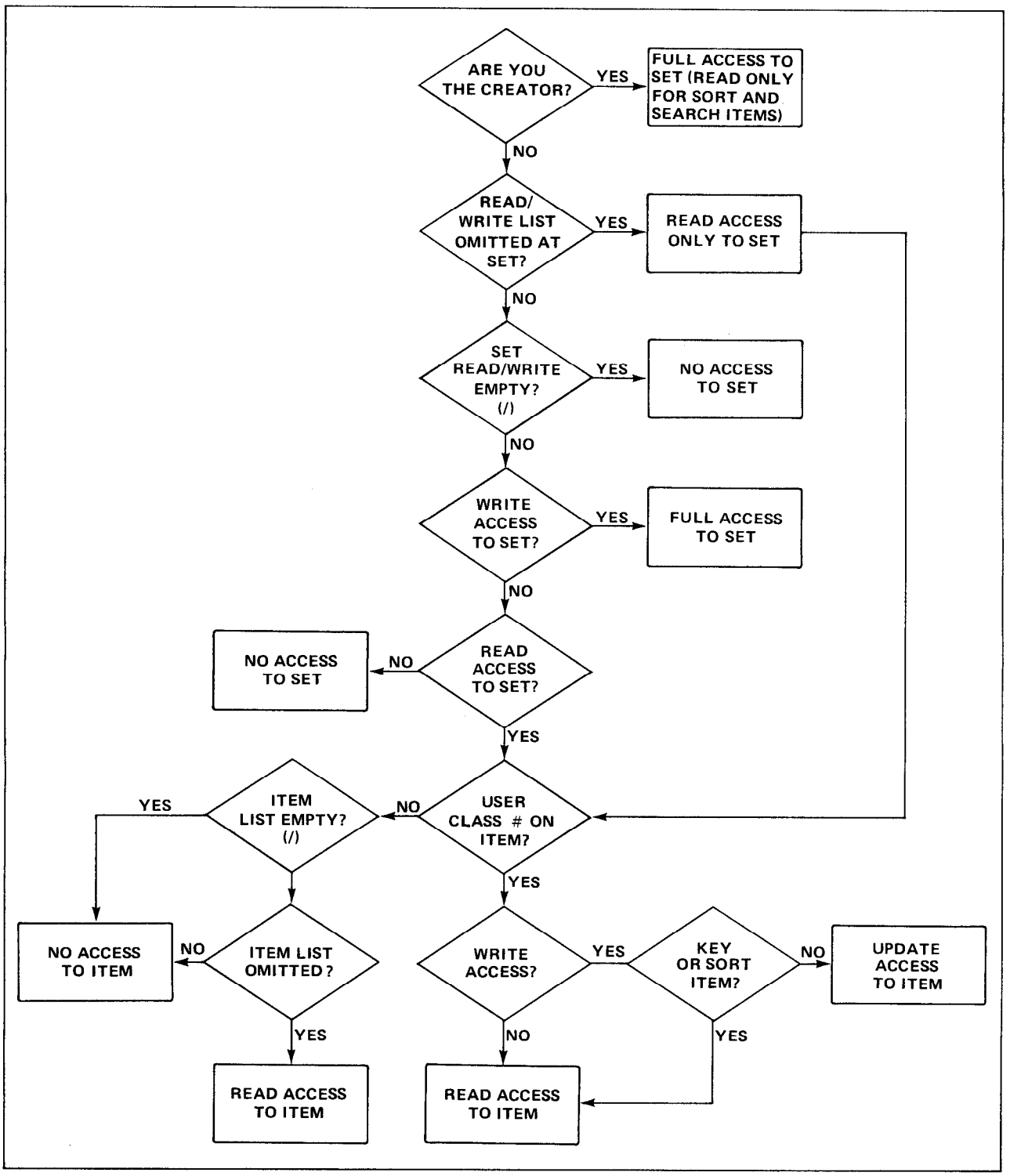


Figure 2-7. Security Flow-Chart

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

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

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

Table 2-4. Sample Read and Write Class Lists

Data Set 1	(0,18,13/9)	Item Read Access	Item Update Access
Data Item A		0, 13, 18, 9	9*
Data Item B	(/13)	13, 9*	13, 9*
Data Item C	(/)	9*	9*
Data Item D	(/9)	9	9
Data Item E	(18/13)	13, 18, 9*	13, 9*
Data Item F	(/13, 18)	13, 18, 9*	13, 18, 9*
Data Item G	(12/0)	0, 9*	0, 9*
Data Item H	(13/)	13, 9*	9*
Data Set 2			
Data Item A		0, 1, . . . , 63	
Data Item I	(13/9)	13, 9	9

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

USER CLASSES AND LOCKING

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

PROTECTION IN RELATION TO LIBRARY PROCEDURES

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

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

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

PROTECTION PROVIDED BY THE IMAGE UTILITIES

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

- They acquire exclusive or semi-exclusive access to the data base being processed. (Section IV contains more information about types of access in the discussion of opening a data base.)

- Only the data base creator or a user supplying the correct *maintenance word* can execute the utilities. The data base creator defines the maintenance word when the data base is created with the DBUTIL utility program. (Refer to Section VIII.) In addition, anyone running the utility programs other than DBRECOV must be logged on to the group in which the data base is catalogued.

- Unrecoverable disc or tape problems are treated as functional failures rather than limited successes and result in program termination.

SECTION III
DEFINING A DATA BASE

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

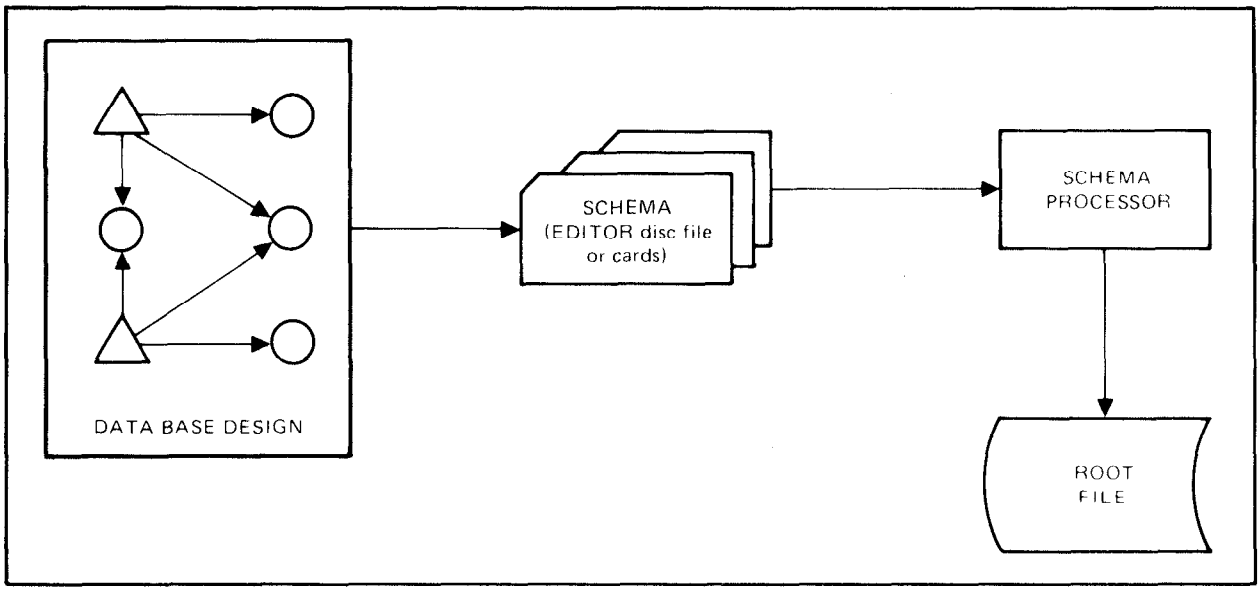


Figure 3-1. Data Base Definition Process

DATA BASE DESCRIPTION LANGUAGE

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

LANGUAGE CONVENTIONS

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

Table 3-1. Additional Conventions

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

SCHEMA STRUCTURE

The overall schema structure is:

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

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

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

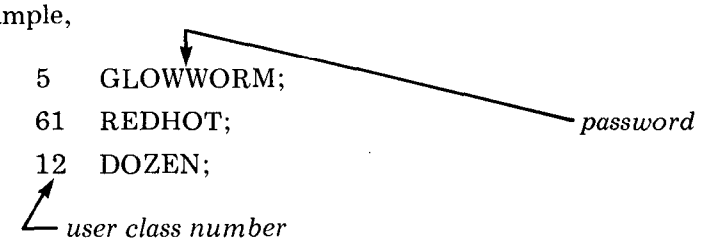
PASSWORD PART

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

The form of the password part is

user class number [*password*] ;
.
.
.
user class number [*password*] ;

For example,



where

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

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

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

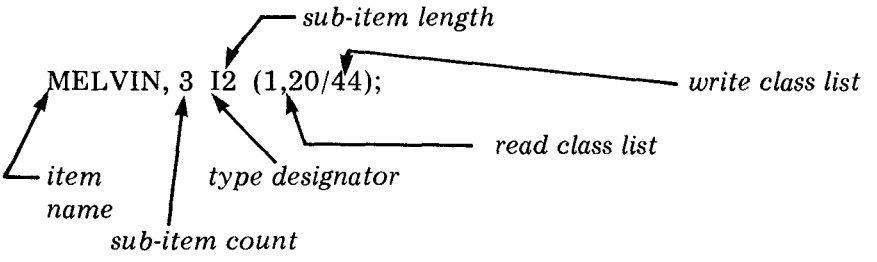
ITEM PART

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

The form of the item part is

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

For example,



where

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

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

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

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

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

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

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

DATA ITEM LENGTH

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

Table 3-2. Type Designators

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

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

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

IMAGE DATA TYPES AND PROGRAM LANGUAGE DATA TYPES

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

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

Table 3-3. IMAGE Type Designators and Programming Languages

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

*Real numbers must have a length of 2 or more words; R and R1 cannot be used by IMAGE.
**BASIC integers cannot have the value -32768.
***Type LOGICAL items >32767 which are accessed as type INTEGER in BASIC programs are treated as negative integers.
****BASIC REAL and LONG data cannot have the value 10^{-78}.

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

DATA ITEMS OF TYPE P

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

- If a value is entered using QUERY, and no sign is specified, the sign is 1111_2.
  If a value is entered using QUERY, and a plus sign is specified, the sign is 1100_2.
  If a value is entered using QUERY, and a minus sign is specified, the sign is 1101_2.

- If a value is entered using a COBOL program, and the PICTURE clause does not specify a sign, the sign is 1111_2.
  If the PICTURE clause specifies a sign and the value is positive, the sign is 1100_2.
  If the PICTURE clause specifies a sign and the value is negative, the sign is 1101_2.

- If a value is entered using an RPG program, a positive or unsigned value's sign is 1100_2 and a negative value's sign is 1101_2.

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

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

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

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

Table 3-4. Examples of an Item Part

```
ITEMS:

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

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

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

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

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

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

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

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

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

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

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

DATA ITEM IDENTIFIERS

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

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

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

The form of the set part for Master Data Sets is

```
                           { MANUAL    }
{NAME:}    set name,       { M         }    [(read class list/write class list)];
{N:   }                    { AUTOMATIC }
                           { A         }

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

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

For example,

```
           set name         read class list        write class list

NAME:      CUSTOMER,MANUAL (1,4/1,8);
ENTRY:     ACCOUNT(1),                             item name
           LAST-NAME,                              path count
           FIRST-NAME,
           INITIAL,
           STREET-ADDRESS,
           CITY,
           STATE,
           ZIP,
           CREDIT-RATING;
CAPACITY:20003;                            maximum entry count
```

where

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

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

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

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

SET PART(MASTERS)

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

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

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

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

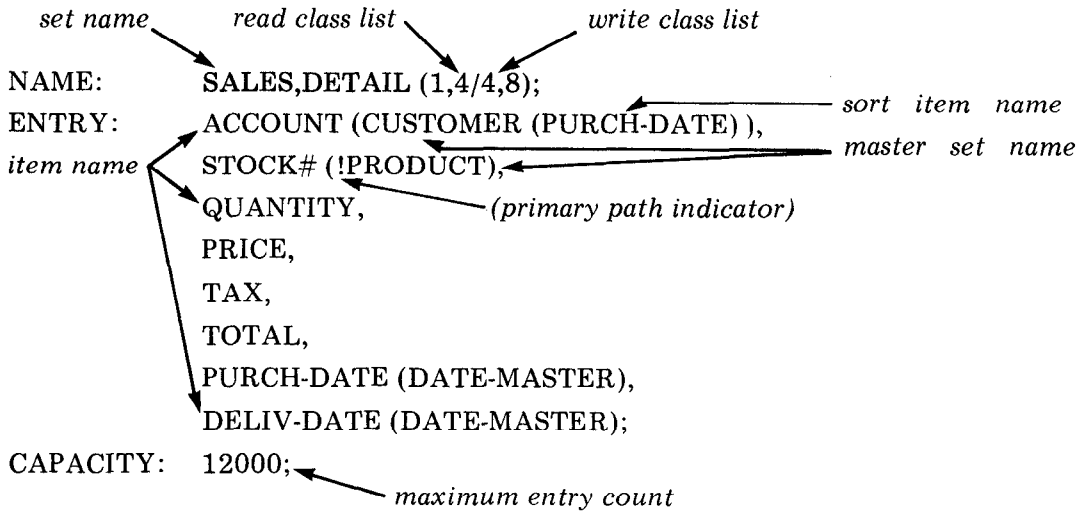
The form of the set part for Detail Data Sets is

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

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

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

For example



where

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

DETAIL or D — denotes a detail data set.

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

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

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

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

SET PART(DETAILS)

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

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

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

MASTER AND DETAIL SEARCH ITEMS

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

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

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

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

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

DATA SET IDENTIFIERS

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

SCHEMA PROCESSOR OPERATION

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

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

:RUN DBSCHEMA.PUB.SYS

to initiate execution of the Schema Processor.

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

Table 3-5. Schema Processor Files

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

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

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

where

n = 1

if an actual file designator has been equated to DBSTEXT

n = 2

if an actual file designator has been equated to DBSLIST

n = 3

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

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

Table 3-6. RUN and FILE Commands, Examples

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

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

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

CREATING THE TEXTFILE

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

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

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

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

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

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

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

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

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

THE DATA BASE CREATOR

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

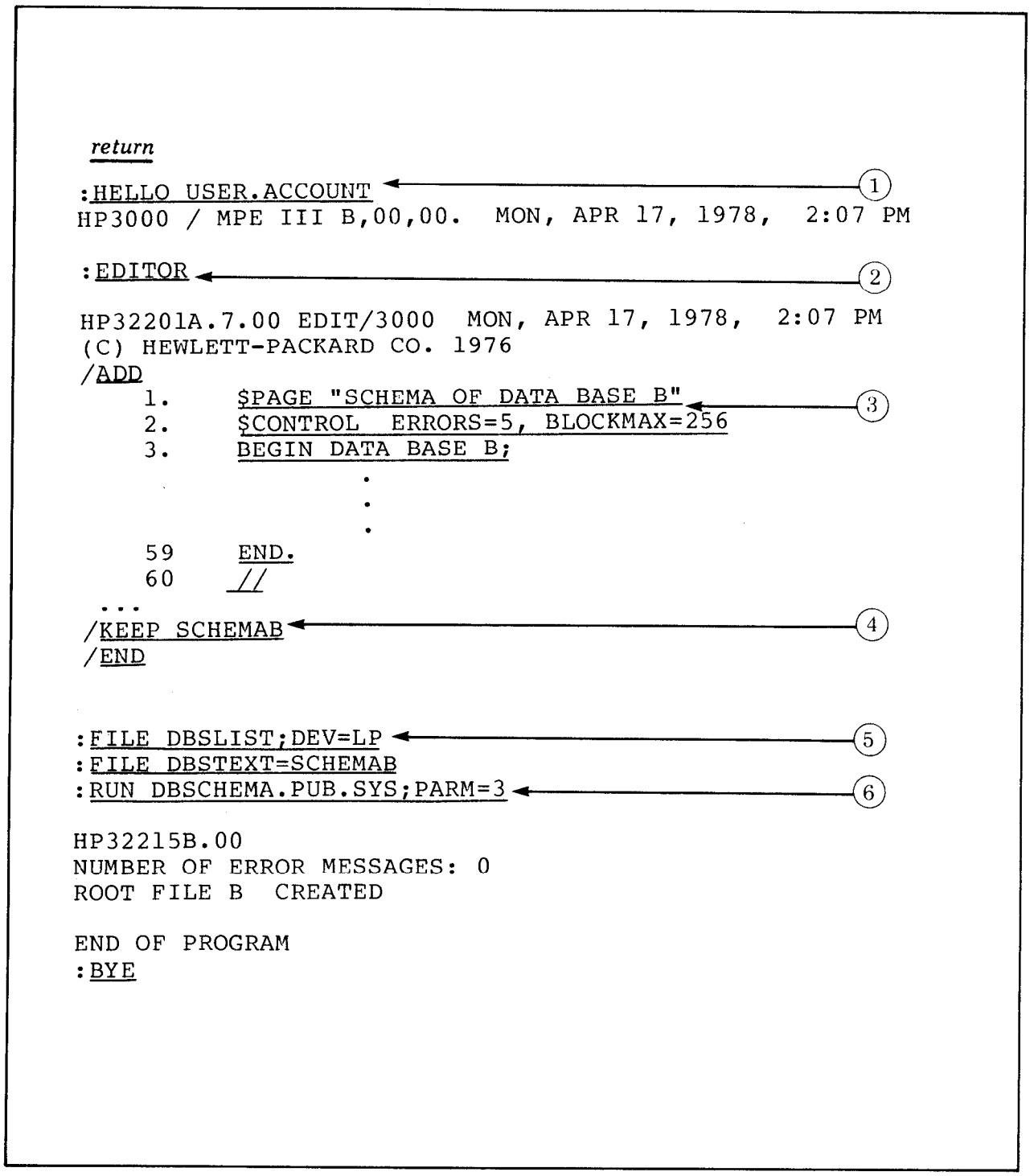


Figure 3-2. Sample Schema Creation Session

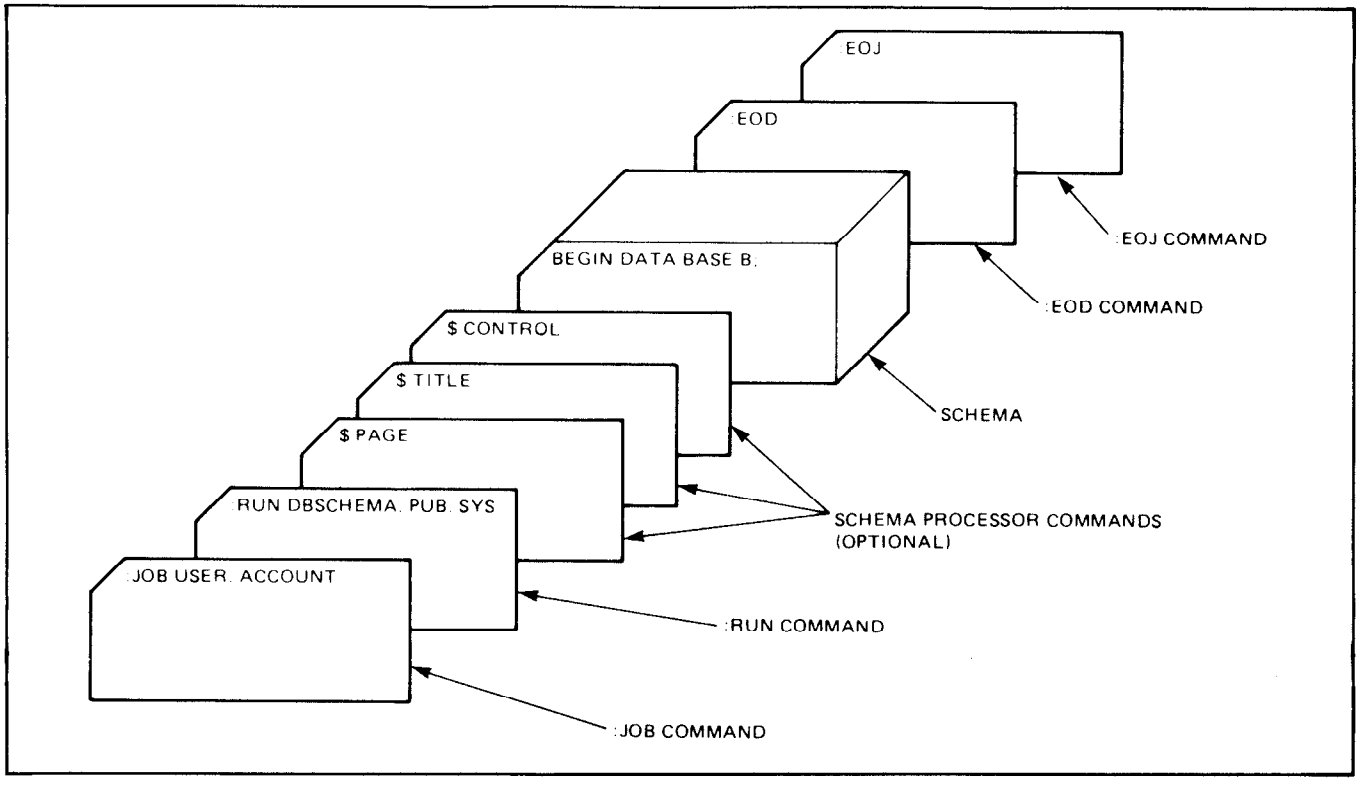


Figure 3-3. Schema Processor Batch Job Stream

SCHEMA PROCESSOR COMMANDS

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

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

Command records may not contain comments.

CONTINUATION RECORDS

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

$PAGE COMMAND

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

The form of the $PAGE command is

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

For example,

$PAGE "STORE DATA BASE SCHEMA", " VERSION 3"

character-string

where

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

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

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

EXAMPLES

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

$PAGE
```

$TITLE

$TITLE COMMAND

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

The form of the $TITLE command is

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

For example,

$TITLE "INVENTORY DATA BASE SCHEMA B. J. BRINDISI"

character-string

where

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

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

EXAMPLE

```
$TITLE""""QUICK""TEST DATA BASE"
```

$CONTROL COMMAND

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

The form of the $CONTROL command is

```
$CONTROL  [LIST]      [,ERRORS=nnn]  [,LINES=nnnnn]  [,ROOT]      [,BLOCKMAX=nnnn]  [,TABLE]
          [NOLIST]                                   [,NOROOT]                      [,NOTABLE]
```

For example,

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

where

LIST	causes each source record of the schema to be printed on the *listfile.*
NOLIST	specifies that only source records with errors be printed on the listfile. An error message is printed after these records.
ERRORS=*nnn*	sets the maximum number of errors to *nnn*. If more than *nnn* errors are detected, the Schema Processor terminates. *nnn* may have a value between 0 and 999, inclusive. The default value is 100.
LINES=*nnnnn*	sets the number of lines per page on the listfile to *nnnnn* which can be between 4 and 32767, inclusive. The default value is 60 if listfile is a line printer and 32767 if it is not one.
ROOT	causes the Schema Processor to create a root file if no errors are detected in the schema.
NOROOT	prevents the Schema Processor from creating a root file.
BLOCKMAX=*nnnn*	sets the maximum physical block length (in words) for any data set in the data base. *nnnn* may have a value between 128 and 2048, inclusive. The default value is 512. This is an important parameter and is discussed in greater detail below.
TABLE	causes the Schema Processor to write a table of summary information about the data sets to the *listfile* device if no errors are detected.
NOTABLE	suppresses the TABLE option.

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

$CONTROL LIST,ERRORS=100,LINES=60, ROOT,BLOCKMAX=512,TABLE

(or 32767)

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

SELECTING THE BLOCK SIZE

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

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

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

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

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

SCHEMA PROCESSOR OUTPUT

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

product identification *product name*

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

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

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

SUMMARY INFORMATION

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

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

  UNREFERENCED ITEMS: *list of items*

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

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

DATA SET NAME	TYPE	FLD CNT	PT CT	ENTR LGTH	MED REC	CAPACITY	BLK FAC	BLK LGTH	DISC SPACE
EMPLOYEE	M	4	1	7	17	500	30	512	72
PROJECT-MASTER	M	2	1	10	20	75	19	382	15
LABOR	D	4	2	10	18	10024	28	506	1436
			TOTAL DISC SECTORS INCLUDING ROOT:						1532

Figure 3-4. Data Set Summary Table

Table 3-7. Data Set Summary Table Information

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

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

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

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

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

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

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

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

  ROOT FILE *data base name* CREATED

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

SCHEMA ERRORS

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

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

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

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

PRECEDING ERRORS — NO ROOT FILE CREATED.

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

SCHEMA PROCESSING TERMINATED.

SCHEMA PROCESSOR EXAMPLE

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

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

$CONTROL LINES=56
BEGIN DATA BASE STORE:

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

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

SETS:

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

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

Figure 3-5. STORE Data Base Schema

```
PAGE 2      STORE


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

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

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

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

END.

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

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


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

ROOT FILE STORE  CREATED.
```

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

BASIC EXAMPLES

To simplify your access to an IMAGE data base through BASIC language programs, it is recommended that you use the BIMAGE interface procedures provided with the IMAGE software. These routines convert all parameter byte addresses to word addresses as required by IMAGE. In addition to calling the necessary IMAGE procedure, the BIMAGE procedures perform the following functions for your convenience:

- automatically pack into a buffer a list of expressions before calling the DBPUT or DBUPDATE procedures
- automatically unpack from a buffer to a list of BASIC variables the values of items returned by DBGET or the values returned by DBINFO
- automatically update the logical length of string variables to which data is transferred from the data base to reflect the length of the string actually transferred.

Table 6-1 lists the BIMAGE interface procedures with the IMAGE procedures to which they correspond. The parameters are described in table 6-2. The corresponding IMAGE procedure parameter is listed next to the BIMAGE parameter.

Table 6-1. BIMAGE Procedure Calls

BIMAGE	CORRESPONDS TO:
XDBOPEN (*B$, W$, mode, status(*)*)	DBOPEN
XDBPUT (*B$,* { *D$* / *d* } *, mode, status(*),* { *L$* / *l* } *, writelist)*	DBPUT
XDBFIND (*B$,* { *D$* / *d* } *, mode, status(*),* { *I$* / *i* } *,* { *A$* / *a* })	DBFIND
XDBGET (*B$,* { *D$* / *d* } *, mode, status(*),* { *L$* / *l* } *, readlist,* { *A$* / *a* })	DBGET
XDBUPDATE (*B$,* { *D$* / *d* } *, mode, status(*),* { *L$* / *l* } *, writelist)*	DBUPDATE
XDBDELETE (*B$,* { *D$* / *d* } *, mode, status(*)*)	DBDELETE
XDBLOCK (*B$,* { *descriplist* } *, mode, status(*)*)	DBLOCK
XDBUNLOCK (*B$,* { *D$* / *d* } *, mode, status(*)*)	DBUNLOCK
XDBCLOSE (*B$,* { *D$* / *d* } *, mode, status(*)*)	DBCLOSE
XDBINFO (*B$,* { *Q$* / *q* } *, mode, status(*), readlist)*	DBINFO
XDBEXPLAIN (*status(*)*)	DBEXPLAIN
XDBERROR (*status(*), M$* [*,length*])	DBERROR

Table 6-2. BIMAGE Procedure Parameters

BIMAGE**	IMAGE	
A$	*argument*	May be any string expression.
a	*argument*	May be a numeric expression or numeric array of any data-type.
B$	*base*	Must be a simple string variable. Value should not be altered between calls to XDBOPEN and XDBCLOSE.
D$	*dset*	May be any string expression.
d	*dset*	May be a type-INTEGER expression.*
descriplist	*qualifier*	Has same form as *writelist*. You should ensure that once BASIC has concatenated the component variables, the result is a valid lock descriptor list (or set name) as defined for DBLOCK. (Parameter ignored for DBLOCK modes 1 and 2)
I$	*item*	May be any string expression.
i	*item*	May be a type-INTEGER expression.*
L$	*list*	May be any string expression or a string array. If it is a string array, all of the string elements are concatenated to form one string whose length may not exceed 255 characters. The concatenated string must form a syntactically correct *list* parameter. Commas must be placed appropriately.
l	*list*	May be an array of type INTEGER.
length	*length*	Must be a simple or subscripted type-INTEGER variable (if not, parameter is ignored.) Parameter is optional but if present, total length of IMAGE message is returned. Value may exceed length of message by BIMAGE procedure if *M$* is too small and message is truncated. Not needed when M$ is a string variable.
M$	*buffer*	Should be a simple or subscripted string variable without substring designators. If message is larger than *M$*, message is truncated on the right. Logical length of *M$* is set to length of message returned by BIMAGE and may not be equal to *length* if message is truncated.
mode	*mode*	Must be type-INTEGER expression.*
Q$	*qualifier*	May be any string expression.
q	*qualifier*	May be a type-INTEGER expression.*
status	*status*	Must be a type-INTEGER array containing at least ten active elements.
W$	*password*	May be any string expression.
readlist	*buffer*	Has form similar to *item list* of BASIC READ or MAT READ statement. May consist of one or more string or numeric simple or subscripted variables or arrays separated by commas. String variables with substring designators and the "FOR-loop" construct are not permitted.
writelist	*buffer*	Has form similar to *item list* of BASIC PRINT or MAT PRINT statement. May consist of one or more string or numeric expressions or arrays separated by commas. "FOR-loop" not permitted. Substring designators are permitted.

* See discussion of type-INTEGER expressions as parameters.

** Note that if you specify an array as a parameter you must obey BASIC syntax rules and append parentheses and asterisks, for example, L$(*,*) or A(*).

Refer to the IMAGE procedure descriptions in Section V for details regarding the purpose of a procedure and its parameters as well as available options.

BIMAGE provides some extensions to the IMAGE procedure calling sequences to simplify your access to the data base:

- BIMAGE allows you to enter a list of expressions in place of the *buffer* parameter. The list is automatically packed into or unpacked from a temporary buffer constructed by the BIMAGE procedures. This facility is also available to construct lock descriptor lists.

- String or numeric expressions are accepted for many parameters. For example, the *dset* parameter may be a string expression when specifying the data set by name or a numeric expression when specifying the data set by number.

STRING VARIABLES

The *physical* length of a string variable determines the number of characters (bytes) read by the XDBGET procedure and the *logical* length of a string variable determines the number of characters written by the XDBPUT and XDBUPDATE procedures. Thus, you should ensure that the physical length of a string variable specified in a DIM or COM statement exactly matches the size of the item to be read by a call to XDBGET.

On the other hand, the same string variable can be used to write items of varying sizes. Substring designators should be used to ensure that the actual string passed to XDBPUT or XDBUPDATE fills the item to be written. For example, if the item is 8 characters long, and substring S$(3) is 2 characters long, S$(3,10) or S$(3;8) fills the item with the S$(3) substring and appends 6 blanks.

If the string variable is an array, the length of each string element or of the concatenated string elements should correspond to the length of the item or sub-item to be written. You can ensure this by specifying substring designators when assigning values to elements of the string array in your BASIC program.

TYPE-INTEGER EXPRESSIONS AS PARAMETERS

Since BASIC treats integral numeric constants as type-REAL, expressions involving constants cannot be passed directly to a type-INTEGER parameter of a BIMAGE procedure. You can define a function such as the following to ensure that a type-INTEGER expression is passed:

```
10 DEF INTEGER FNI(X)=X
```

When a procedure call is made, the function is used in this way:

```
50 CALL XDBLOCK(B$, D$, FNI(expression),S(*) )
```

The function FNI converts *expression* to type-INTEGER.

THE READLIST, WRITELIST, AND DESCRIPLIST PARAMETERS

When specifying string expressions in a *readlist*, *writelist*, or *descriplist*, each string expression should correspond to a data item or sub-item, or groups of items or sub-items in the case of string arrays. You should not specify several string expressions as the source or destination of one item or sub-item. The transfer of strings to or from the data base always begins on a word boundary of the buffer. Therefore, writing from or reading into two odd-length strings is not the same as writing or reading into one even-length string.

THE STATUS PARAMETER

If the *status* parameter is a type-INTEGER variable, a condition word is returned in the first word and the second word is set to zero if *status* is at least a two-element array. The condition word will have a value equal to those listed for the corresponding IMAGE procedure and, in addition, may contain one of the conditions listed in table 6-3.

If the *status* parameter is not type-INTEGER, the BIMAGE procedures cannot return a condition word for the common error: failure to declare the *status* variable type-INTEGER. This error will usually result in the BASIC message UNDEFINED VALUE the first time the *status* array contents are examined.

Table 6-3. Additional BIMAGE Condition Word Values

EXCEPTIONAL CONDITIONS:		PROCEDURES:
51	Insufficient stack for temporary buffer.	XDBGET,XDBPUT, XDBINFO,XDBUPDATE
52	Invalid number of parameters.	All procedures except XDBERROR and XDBEXPLAIN
53	Invalid parameter.	
54	*status* array has less than 10 elements.	

OPEN DATA BASE

```
10 DIM B$[8],P$[8]
20 INTEGER S[10],M
30 B$="  STORE;"
40 INPUT "ENTER PASSWORD: ",P$[1;8]
50 INPUT "ENTER ACCESS MODE (1-8):  ",M
60 CALL XDBOPEN(B$,P$,M,S[*])
70 IF S[1]<>0 THEN 9300
              .
              .
              .
      (code to use data base)
              .
9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBEXPLAIN(S[*])
9320 STOP
              .
              .
              .
```

In this example, the STORE data base is opened in the access mode entered by the user and with a user class number corresponding to the password entered by the user and stored in the P$ string. If the password is less than 8 characters the P$ string is padded with blanks. The first word of the status array, S, is tested to determine whether the procedure executed successfully. If not, an error message is printed.

ADD ENTRY

```
 10 DIM B$[8],P$[8],A$[8],C$[20]
 20 INTEGER S[10],M,M1
 30 B$="  STORE;"
 40 INPUT "ENTER PASSWORD: ",P$[1;8]
 50 INPUT "ENTER ACCESS MODE (3,4): ",M
 60 GOTO M OF 70,70,90,90
 70 PRINT "CANNOT ADD ENTRIES IN THIS ACCESS MODE"
 80 GOTO 50
 90 CALL XDBOPEN(B$,P$,M,S[*])
100 IF S[1]<>0 THEN 9300
110 INPUT "ENTER STOCK# OR / TO TERMINATE: ",A$[1;8]
120 IF A$[1,1]="/" THEN GOTO 9900
130 INPUT "ENTER DESCRIPTION: ",C$[1;20]
140 M1=1
150 CALL XDBPUT(B$,"PRODUCT;",M1,S[*],"@;",A$[1;8],C$[1;20])
160 IF S[1]<>43 THEN 190
170 PRINT "DUPLICATE STOCK NUMBER"
180 GOTO 110
190 IF S[1]<>16 THEN 220
200 PRINT "DATA SET FULL"
210 GOTO 9900
220 IF S[1]<>0 THEN 250
230 PRINT "NEW PRODUCT HAS BEEN ENTERED"
240 GOTO 110
250 IF S[1]=-23 THEN 290
260 PRINT "DBPUT FAILURE"
270 CALL XDBEXPLAIN(S[*])
280 GOTO 9900
290 PRINT "YOUR PASSWORD DOES NOT ALLOW YOU TO ADD ENTRIES"
300 GOTO 9900
```

After data base opened, first word of B$ contains data segment number.

```
              :
              :
```

9300 (code same as example above)

9900 (close data base)

This sample code adds an entry to the PRODUCT manual master data set. Note that the B$ string used to open the data base is the *base* parameter in this call. It should not be changed after the call to XDBOPEN since this call saves a data segment number in the first word of B$. The list of items to be added is specified as @; which indicates that values are specified for all items in the entry. The values for the STOCK# and DESCRIPTION data items are stored in A$ and C$. Sample values are "7474Z74Z" and "ORANGE CRATEΔΔΔΔΔΔΔΔ".

In the example, the condition word of the status array is tested for a value of 43, indicating that an entry with the specified STOCK# search item value already exists in the data set, or 16, indicating that the data set is full, or –23, indicating that the user's password does not grant write access to the data set.

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

READ ENTRY (SERIALLY)

```
 10 DIM B$[8],P$[8],D1$[14],L1$[20],S1$[16],S2$[2]
 20 INTEGER S[10],M,M1,M2
 30 B$="  STORE;"
 40 M1=1
 50 INPUT "ENTER PASSWORD: ",P$[1;8]
 60 INPUT "ENTER ACCESS MODE (1-8): ",M
 70 CALL XDBOPEN(B$,P$,M,S[*])
 80 IF S[1]<>0 THEN 9300
200 M2=2
210 DI$="SUP-MASTER;"
220 LI$="SUPPLIER,STATE;"  <-------------------- readlist
230 CALL XDBGET(B$,D1$,M2,S[*],L1$,S1$,S2$,"")
240 IF S[1]<>11 THEN 270
250 GOSUB 900
260 GOTO 230
270 IF S[1]<>0 THEN 320
280 PRINT "SUPPLIER= ",S1$,"STATE= ",S2$
290 INPUT "CONTINUE (Y OR N)? ",X$
300 IF X$[1,1]="Y" THEN GOTO 230
310 GOTO 9900
320 IF S[1]=-21 THEN 360
330 PRINT "DBGET FAILURE"
340 CALL XDBEXPLAIN(S[*])
350 GOTO 9900
360 PRINT "YOU DO NOT HAVE ACCESS TO THIS DATA"
370 GOTO 9900
          •
900 (routine to rewind data set)


9300 (same as XDBOPEN example)
          •
          •
9900 (close data base)
```

To read the next entry of the SUP-MASTER data set, a mode of 2 is used. This directs the XDBGET (and DBGET) procedure to perform a forward serial read. In the example, the list in the L1$ string specifies two data items to be read. After returning to the calling program, the S1$ string contains the STOCK# data item value and S2$ contains the DESCRIPTION data item value. The *argument* parameter is ignored if *mode* equals 2, therefore, a null string may be used for this parameter.

If an end-of-file is encountered the condition word is set to 11. In this case, if the user wants to continue, the routine rewinds the data set and tries the read again. A rewind routine is shown later in the examples of the XDBCLOSE procedure. The rewind reinitializes the current record pointer so that the next request for a forward serial read will read the first entry in the data set.

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

READ ENTRY (CALCULATED)

```
 10 DIM B$[8],P$[8],C$[20],S0$[8]
 20 INTEGER S[10],M1,M
 30 B$="  STORE;"
 40 M1=1
 50 DEF INTEGER FNI(X)=X
 60 INPUT "ENTER PASSWORD: ",P$[1;8]
 70 INPUT "ENTER ACCESS MODE (1-8): ",M
 80 CALL XDBOPEN(B$,P$,M,S[*])
 90 IF S[1]<>0 THEN 9300
300 INPUT "ENTER STOCK# OR / TO TERMINATE: ",S0$[1;8]
310 IF S0$[1,1]="/" THEN GOTO 9900
320 CALL XDBGET(B$,"PRODUCT ",FNI(7),S[*],"DESCRIPTION;",C$,S0$)
330 IF S[1]<>17 THE GOTO 360
340 PRINT "NO SUCH STOCK NUMBER"
350 GOTO 300
360 IF S[1]=0 THEN GOTO 410
370 IF S[1]=-21 THEN 430
380 PRINT "DBGET FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 PRINT S0$,C$
420 GOTO 300
430 PRINT "YOUR PASSWORD DOES NOT GRANT ACCESS TO DATA REQUESTED"
440 GOTO 9900
      :
9300(same code as XDBOPEN example)
      .
9300(close data base)
      :
```

To locate the PRODUCT data set entry which has a STOCK# search item value equal to the one entered in S0$ by user, a calculated read is used. The mode is 7 and the item to be read is DESCRIPTION. After XDBGET returns control to the calling program, the description is in C$. If no entry exists with the specified STOCK# value, the condition word is 17. If the user does not have read access to the requested data, a condition word of -21 is returned.

READ ENTRY (BACKWARD CHAIN)

```
BITST5
 10 DIM B$[8],P$[8],I1$[6],A$[8],A1$[16]
 20 INTEGER S[10],M1,M,M6
 30 B$="  STORE;"
 40 M1=1
 50 M6=6
 60 INPUT "ENTER PASSWORK: ",P$[1;8]
 70 INPUT "ENTER ACCESS MODE (1-8): ",M
 80 CALL XDBOPEN(B$,P$,M,S[*])
 90 IF S[1]<>0 THEN 9300
300 INPUT "ENTER LASTSHIPDATE (YYMMDD) OR E TO EXIT: ",I1$[1;6]
310 IF I1$[1,1]="E" THEN GOTO 9900
320 CALL XDBFIND(B$,"INVENTORY ",M1,S[*],"LASTSHIPDATE;",I1$)
330 IF S[1]<>17 THEN GOTO 360
340 PRINT "NO SHIPMENTS ON THAT DATE"
350 GOTO 300
360 IF S[1]=0 THEN GOTO 410
370 IF S[1]=-21 OR S[1]=-52 THEN 480
380 PRINT "DBFIND FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 CALL XDBGET(B$,"INVENTORY;",M6,S[*],"STOCK#,SUPPLIER;",A$,A1$"")
420 IF S[1]<>14 THEN GOTO 450
430 PRINT "NO MORE SHIPMENTS ON THIS DATE"
440 GOTO 300
450 IF S[1]<>0 THEN GOTO 500
460 PRINT A$,A1$
470 GOTO 410
480 PRINT "YOUR PASSWORD OR ACCESS MODE DOES NOT GRANT ACCESS TO DATA"
490 GOTO 9900
500 PRINT "DBGET FAILURE"
510 GOTO 390
```

•
•
•

9300 (same as XDBOPEN example)

•
•
•

9900 (close data base)

First the XDBFIND procedure is called to determine the location of the first and last entries in the chain. The call parameters include the detail data set name, the name of the detail search item used to define a path with the DATE-MASTER data set, and the search item value of both the master entry containing the chain head and the detail entries making up the chain. The search item value is requested from the user and stored in I1$, for example, the user may enter 760314.

If no entry in the DATE-MASTER has a search item value entered, the condition word will be 17. If the user does not have read access to the data, a condition word of -21 or -52 is returned.

If the XDBFIND procedure executes successfully, a call to the XDBGET procedure with a mode parameter of 6 reads the last entry in the chain. Subsequent calls to XDBGET with the same mode read backward through the chain until the first entry has been read. If the condition word is 14, the beginning of the chain has been reached and no more entries are available, or there are no entries in the chain.

If an entry is successfully read, the program uses the STOCK# value stored in A$ and the SUPPLIER value stored in A1$ and then returns to statement 350 to read another entry in the chain.

UPDATE ENTRY

```
 10 DIM B$[8],P$[8],D1$[12],I2$[16],A5$[26],S9$[16]
 20 INTEGER S[10],M
 30 B$="  STORE;"
 40 DEF INTEGER FNI(X)=X
 50 D1$="SUP-MASTER "
 60 I2$="STREET-ADDRESS;"
 70 INPUT "ENTER PASSWORD: ",P$[1;8]
 80 M=3
 90 CALL XDBOPEN(B$,P$,M,S[*])
100 IF S[1]<>0 THEN 9300
200 INPUT "ENTER SUPPLIER OR / TO TERMINATE: ",S9$[1;16]
210 IF S9$[1,1]="/" THEN GOTO 9900
220 CALL XDBGET(B$,D1$,FNI(7),S[*],I2$,A5$,S9$)
230 IF S[1]=-21 THEN GOTO 290
240 IF S[1]=0 THEN GOTO 310
250 IF S[1]=17 THEN GOTO 430
260 PRINT "DBGET FAILURE"
270 CALL XDBEXPLAIN(S[*])
280 GOTO 9900
290 PRINT &
    "YOUR PASSWORD OR ACCESS MODE DOES NOT ALLOW ACCESS TO THIS DATA"
300 GOTO 9900
310 PRINT "CURRENT ADDRESS: ",A5$
320 INPUT "ENTER NEW ADDRESS: ",A5$[1;26]
330 CALL XDBUPDATE(B$,D1$,FNI(1),S[*],I2$,A5$)
340 IF S[1]<>42 THEN GOTO 370
350 PRINT "YOU ARE NOT ALLOWED TO ALTER THIS ITEM"
360 GOTO 200
370 IF S[1]=0 THEN 410
380 PRINT "DBUPDATE FAILURE"
390 CALL XDBEXPLAIN(S[*])
400 GOTO 9900
410 PRINT "ADDRESS CHANGED"
420 GOTO 200
430 PRINT "NO SUCH SUPPLIER"
440 GOTO 200
          •
9310 (same as XDBOPEN example)
          :
9900 (close data base)
```

Before an entry can be updated it must be located. In this example, the entry is located with a calculated XDBGET that reads the STREET-ADDRESS item in the SUP-MASTER data set. The entry is located by using the SUPPLIER search item with a value supplied by the user. If the read is successful, the current address is printed and the application program user is prompted for the new address which is moved into A5$. The XDBUPDATE procedure is then called to alter the STREET-ADDRESS data item in the entry.

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

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

DELETE ENTRY (WITH LOCKING AND UNLOCKING)

```
 10 DIM B$[8],P$[8],D1$[12],S9$[16],A5$[16]
 20 INTEGER S[10],M2,M1,M4
 30 B$="  STORE;"
 40 DEF INTEGER FNI(X)=X
 50 D1$="SUP-MASTER "
 60 INPUT "ENTER PASSWORD: ",P$[1;8]
 70 M1=1
 80 M2=2
 85 M4=4
 90 CALL XDBOPEN(B$,P$,M1,S[*])
100 IF S[1]<>0 THEN 9300
110 INPUT "ENTER SUPPLIER OR / TO TERMINATE: ",S9$[1;16]
120 IF S9$[1,1]="/" THEN GOTO 9900
130 CALL XDBLOCK(B$,D1$,M4,S[*])
140 IF S[1]<=0 THEN 170
150 PRINT "DATA SET IS BUSY. TRY AGAIN LATER."
160 GOTO 9900
170 IF S[1]=0 THEN 210
180 PRINT "DBLOCK FAILURE"
190 CALL XDBEXPLAIN(S[*])
200 GOTO 9900
210 CALL XDBGET(B$,D1$,FNI(7),S[*],"SUPPLIER;",A5$,S9$)
220 IF S[1]=0 THEN 330
230 IF S[1]=-21 THEN 280
240 IF S[1]=17 THEN 310
250 PRINT "DBGET FAILURE"
260 CALL XDBEXPLAIN(S[*])
270 GOTO 290
280 PRINT "YOUR PASSWORD DOES NOT GRANT ACCESS TO DATA SET"
290 GOSUB 9000
300 GOTO 9900
310 PRINT "NO SUCH SUPPLIER"
320 GOTO 430
330 CALL XDBDELETE(B$,D1$,FNI(1),S[*])
340 IF S[1]<>44 THEN GOTO 370
```

```
 350 PRINT "INVENTORY ENTRIES EXIST, SUPPLIER CANNOT BE DELETED"
 360 GOTO 430
 370 IF S[1]=0 THEN GOTO 420
 380 IF S[1]=-23 THEN 280
 390 PRINT "DBDELETE FAILURE"
 400 CALL XDBEXPLAIN(S[*])
 410 GOTO 9900
 420 PRINT "SUPPLIER DELETED"
 430 GOSUB 9000
 440 GOTO 110
9000 CALL XDBUNLOCK(B$,"",M1,S[*])
9010 IF S[1]=0 THEN RETURN
9020 PRINT "DBUNLOCK FAULURE"
9030 CALL XDBEXPLAIN(S[*])
9040 GOTO 9900
9300 PRINT "DBOPEN FAILURE"
9310 CALL XDBEXPLAIN(S[*])
9320 STOP
9900 CALL XDBCLOSE(B$,"",FNI(1),S[*])
9910 IF S[1]=0 THEN STOP
9920 PRINT "DBCLOSE FAILURE"
9930 GOTO 9310
9999 END
```

In the example above, the program calls XDBLOCK to lock the SUP-MASTER data set. Since mode 4 is used, the program must check the condition word when DBLOCK returns control to verify that the data base is locked and the calling program has exclusive access. If this is so, the condition word is 0.

If the data is successfully locked, the program performs the necessary data base operations. In this case, it deletes an entry. Before the entry can be deleted, the current record of the data set must be that of the entry to be deleted. This record may be located by calling XDBGET. The program may request the name of the supplier whose record is to be deleted and use XDBGET in calculated mode to locate the appropriate entry. If entries in the INVENTORY data set exist that have the same SUPPLIER value as the entry to be deleted, the condition word is set to 44 and the entry is not deleted.

After the entry is deleted the data set is unlocked by XDBUNLOCK.

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

REQUEST DATA SET INFORMATION

```
 10 DIM B$[8],P$[8]
 20 INTEGER S[10],D2[7],M
 30 B$="  STORE;"
 40 INPUT "ENTER PASSWORD: ",P$[1;8]
 50 INPUT "ENTER ACCESS MODE (1-8): ",M
 60 CALL XDBOPEN(B$,P$,M,S[*])
 70 IF S[1]<>0 THEN 9300
300 M=203
310 CALL XDBINFOR(B$,"",M,S[*],D2[*])
320 IF S[1]=0 THEN 350
330 CALL DBEXPLAIN(S[*])
340 GOTO 9900
350 PRINT "YOU HAVE ACCESS TO";D2[1];"DATA SETS AS FOLLOWS:"
360 FOR I=2 TO D2[1]+1
370   PRINT D2[I]
380 NEXT I
390 GOTO 9900
          •
9300 (same as XDBOPEN example)
9900 (close data base)
```

The procedure call in this example obtains the numbers of data sets that are available to the current user class by specifying mode 203. If the user class number is 12 and the procedure executes successfully, the D2 array contains:

D2(1)	4	Access to 3 data sets.
D2(2)	2	Read access to data set 2.
D2(3)	-3	Modify access to data set 3
D2(4)	-5	and data set 5.
D2(5)	6	Read and possibly update access to data set 6.

REWIND DATA SET

```
 10 DIM B$[8],P$[8],D1$[14],L1$[20],S1$[16],S2$[2]
 20 INTEGER S[10],M,M1,M2
 30 B$="  STORE;"
 40 M1=1

    (open data base)

210 D1$="SUP-MASTER;"
          •
    (read data set serially)

900 INTEGER M3
910 M3=3
920 CALL XDBCLOSE(B$,D1$,M3,S[*])
930 IF S[1]=0 THEN RETURN
940 PRINT "DBCLOSE FAILURE"
950 CALL XDBEXPLAIN(S[*])
960 GOTO 9900

9900 (close data base)
```

To rewind the SUP-MASTER data set, a call to DBCLOSE is made with mode equal to 3. The dynamic status information in the Data Set Control Block for SUP-MASTER is reset, including the current record number. If a serial read request encounters an end-of-file, this call resets the current record to the beginning of the data set and another serial read request reads the first entry in the data set.

CLOSE DATA BASE

```
  10 DIM B$[8],P$[8]
  20 INTEGER S[10],M
  30 B$="  STORE;"
  40 DEF INTEGER FNI(X)=X

           •
           •
           •

9900 CALL XDBCLOSE(B$,"",FNI(1),S[*])
9910 IF S[1]=0 THEN STOP
9920 PRINT "DBCLOSE FAILURE"
9930 GOTO 9310
9999 END
```

This call closes the data base. It is issued after the program has completed all data base operations and before program termination.

PRINT ERROR

```
   10 DIM B$[8]
   20 INTEGER S[10]
          :
 9310 CALL XDBEXPLAIN(S[*])
 9320 STOP
```

A call to DBEXPLAIN prints a message on the $STDLIST device which interprets the contents of the status array, S. This is the routine which is called to display the status in the preceding examples.

MOVE ERROR TO BUFFER

```
   10 DIM B$[8],P$[8],M$[72]
   20 INTEGER S[10],M,M1
   30 B$="  STORE;"
   40 M1=1
   50 INPUT "ENTER PASSWORD: ",P$[1;8]
   60 INPUT "ENTER ACCESS MODE (1-8): ",M
   70 CALL XDBOPEN(B$,P$,M,S[*])
   80 IF S[1]<>0 THEN 9300
   90 PRINT "DATA BASE OPENED"
  100 GOTO 9900
 9300 PRINT "DBOPEN FAILURE"
 9310 CALL XDBERRORS(S[*],M$)
 9320 PRINT M$
 9330 STOP
```

In this example, a call to DBERROR returns one of the messages appropriate to the current condition word. For example, if the condition word is equal to 16, the message returned in M$ is THE DATA SET IS FULL. Note that the length parameter need not be included since the logical length of M$ is set by XDBERROR.