Xerox Extended Data Management System (EDMS)

Sigma 6/7/9 Computers

Reference Manual

90 30 12C

XEROX

Xerox Extended Data Management System (EDMS)

Xerox 560 and Sigma 6/7/9 Computers

Reference Manual

90 30 12C
90 30 12C-1

June 1975

File No.: 1X33
XP82, Rev. 0
Printed in U.S.A.

NOTICE

This publication is a revision of the Xerox Extended Data Management System (EDMS) Reference Manual 90 30 12C. This revision incorporates the Revision Package dated June 1975. A change in the text from that of the previous manual is indicated by a vertical line at the margin of the page. EDMS provides all of the features of Basic DMS plus additional features.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 9 Computer/Reference Manual	90 17 33
Xerox Sigma Glossary of Computer Terminology	90 09 57
Xerox Control Program-Five CP-V/TS Reference Manual	90 09 07
Xerox Control Program-Five CP-V/OPS Reference Manual	90 16 75
Xerox Control Program-Five CP-V/TS User's Guide	90 16 92
Xerox ANS COBOL/LN Reference Manual	90 15 00
Xerox ANS COBOL (BPM)/OPS Reference Manual	90 15 01
Xerox Extended FORTRAN IV/LN Reference Manual	90 09 56
Xerox Extended FORTRAN IV/OPS Reference Manual	90 11 43
Xerox Meta-Symbol/LN, OPS Reference Manual	90 09 52
Xerox Data Management System (DMS)/Reference Manual	90 17 38
Xerox Extended Data Management System (EDMS)/User's Guide	90 30 37
Xerox Interactive Database Processor (IDP)/LN, OPS Reference Manual	90 30 66
Xerox APL/LN, OPS Reference Manual	90 19 31

Manual Content Codes: BP - batch processing, LN - language, OPS - operations, RP - remote processing, RT - real-time, SM - system management, TS - time-sharing, UT - utilities

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory.

CONTENTS

APPENDIXES

FIGURES

TABLES

1. INTRODUCTION

The Xerox Extended Data Management System (EDMS) operates on Sigma 6/7/9 and Xerox 560 computers under the control of the Xerox Control Program-Five (CP-V), and in conjunction with COBOL, Meta-Symbol, FORTRAN applications programs or the APL processor. It is designed specifically for use by organizations that require the same data to be used for many purposes and by many different applications programs.

Extended DMS provides a capability for accumulating large volumes of data into a single database, which may be structured to reflect any desired data relationships. The structuring and related concepts are explained in Chapter 2, "Extended DMS Overview".

A special Extended DMS processor, the File Definition Processor (DMSFDP), creates a database description in two phases. The first phase generates a schema file that describes the complete database, its file size requirements, storage and retrieval techniques, privacy controls, etc. In the second phase, the DMSFDP creates the subschema file by extracting information from the schema file. The subschema may describe the complete database or only those portions that are required by a specific application. The DMSFDP, its Data Definition Language (DDL) input, and its operational interface with CP-V are explained in Chapter 3.

The Database Manager (DBM) consists of a number of library routines, which are explained in Chapter 4. Included in the explanation are the routine call formats for COBOL, Meta-Symbol, and FORTRAN, and descriptions of error processing, journaling, tracing, and statistics collection. Also included are instructions for loading applications programs with the library routines under CP-V. The APL/EDMS interface is described in Chapter 8.

The Extended DMS Utility processors (DMSINIT, DMSDUMP, DMSLOAD, and DMSSUMS) are described in Chapter 5. The use of these processors for initializing files, saving and restoring the database, and printing summary statistics is explained. Also explained are the operational interfaces of these processors with CP-V.

The Extended DMS Restructuring processors (DMSANLZ and DMSREST) are described in Chapters 6 and 7, respectively. The requirements analysis function performed by DMSANLZ and the restructuring function performed by DMSREST for the purpose of altering an existing database, are explained.

2. EXTENDED DMS OVERVIEW

The Extended Data Management System (EDMS) serves an an interface between a user and his data. The user defines his database and generates applications programs that communicate with EDMS in terms of the defined data characteristics and relationships. EDMS, in turn, communicates with the host operating system in terms of files, granules, etc., to transfer the specified data values to and from the database in response to user program requests.

The concept of a database is central to the design of EDMS. An EDMS database is an organized, interrelated collection of information required for various types of activities (e.g., a company's accounting, inventory, and personnel records). Its purpose is to make the same information available for many different uses without incurring the overhead of redundant storage. The value of an EDMS database is realized when there is a need to access the same data values in several different ways, for several different purposes. For example, purchase order data may be used by both accounts payable and inventory control. Accounts payable may need all data for all purchase orders to each vendor. Inventory control may need the total number of parts ordered from all vendors for each type of part ordered. To reduce the number of times the counts of parts ordered must be stored or to reduce the number of times a file must be sorted to produce the information in the desired order, purchase-order data may be stored in an EDMS database and simply linked in the desired ways. Similarly, information on, for example, students assigned to a particular class may be linked in several different ways for use in generating class rosters and in generating student grade reports.

The EDMS capability for accommodating multiple relationships among data values in a database is the most important aspect of the system. Data relationships are described in the following paragraphs along with the system features provided for managing the database, the physical structure of the database files, and the facilities available for database restructuring.

Data Relationships

The term "network-structured" refers to the relationships that can exist in an EDMS database. It implies that a unit of data may be associated with more than one other data unit. For example, information specifying parts on order can be associated with information describing the vendors from whom the parts were ordered, and with stock information on the parts. Relationships in an EDMS database are described in terms of items, groups, and sets.

An item is a logical construct that defines the characteristics of a number of similar data values. The concept of an item is analogous to that of a field. An item occurrence is a single data value with the specified characteristics. For example, Smith might be an occurrence of an item called LASTNAME.

A group is a logical construct that defines a number of similar collections of item occurrences. A group occurrence includes a fixed number of item occurrences, each in a fixed position relative to the others. For example, an occurrence of a group called EMPLOYEE might include an occurrence of the item LASTNAME, an occurrence of the item FIRSTNAME, and an occurrence of the item EMPLOYEENUMBER. Two group occurrences could be depicted as

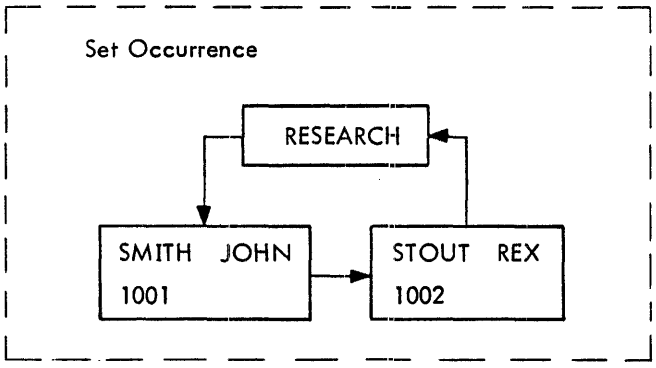
SMITH	JOHN
1001	etc.

STOUT	REX
1002	etc.

A group occurrence can be considered as analogous to a record and the group itself to a record description or definition.

A set is a logical construct that defines and controls the links existing between occurrences of specified groups. A set occurrence consists of one occurrence of the group defined as owner, plus zero, one or more occurrences of the group (or groups) defined as members. For example, a DEPARTMENT group, with an item DEPT-NAME could be

defined as the owner of DEPT-PERSONNEL set. If John Smith and Rex Stout were the only two employees in the research department and EMPLOYEE the only group defined as a member of DEPT-PERSONNEL set, an occurrence of the set could be depicted as follows:



A set occurrence is also somewhat similar to a record, in the sense that it contains all of a certain type of information about an entity (the names of all employees in a department, in the example above).

The links defining a set occurrence are established between the one owner occurrence and the member occurrences, if any. A notation such as shown in Figure 1 can be used to depict the relationships that exist between the one owner group occurrence and the member group occurrences in each occurrence of the set. It should be noted that Figure 1 shows a shorthand notation in which each box may represent many data values, and each connecting line may represent many different set occurrences, each consisting of one owner group occurrence and zero, one, or many member group occurrences.

Given these cautions, we can then describe groups as being owners or members of sets, and a set as consisting of one owner group and one or more member groups. A group can participate in one or more sets as owner and one or more sets as a member. For example, the group named GROUP-2 in Figure 1 is a member of the set named SET-A and the owner of the sets named SET-B and SET-C.

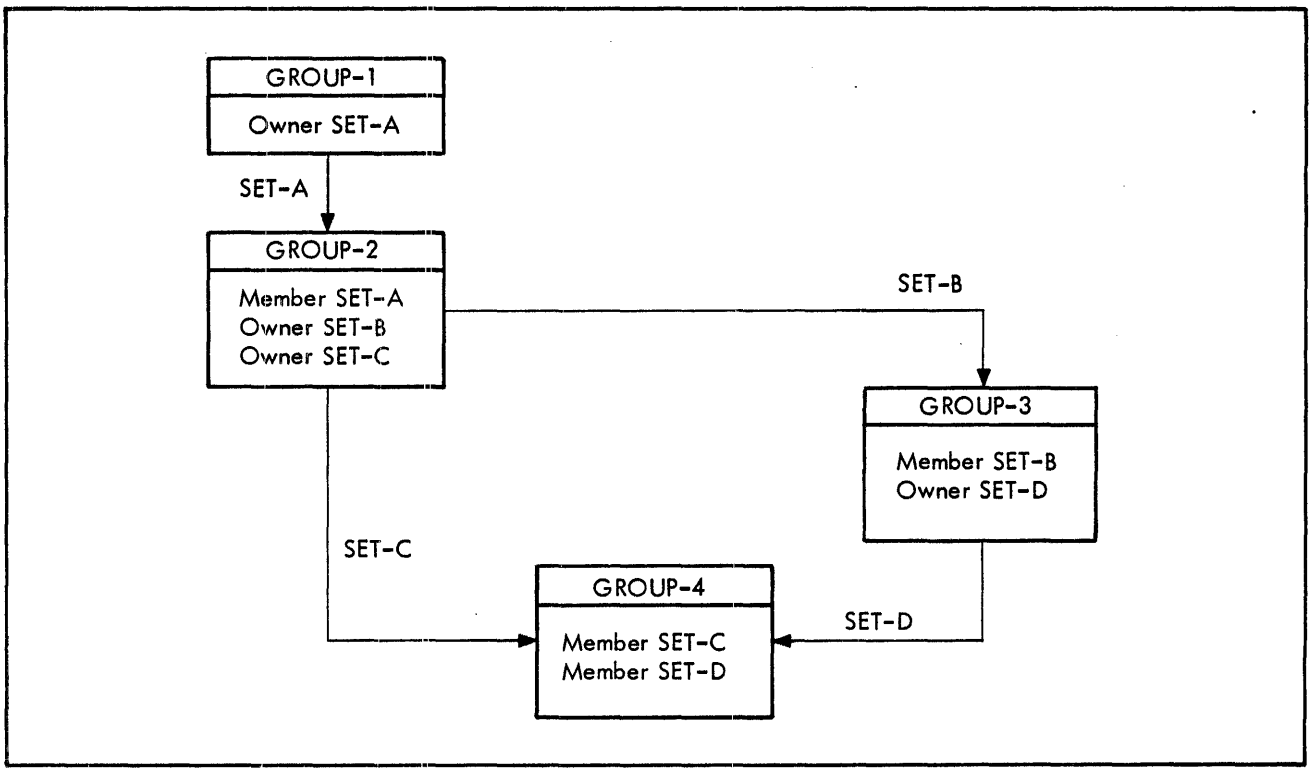
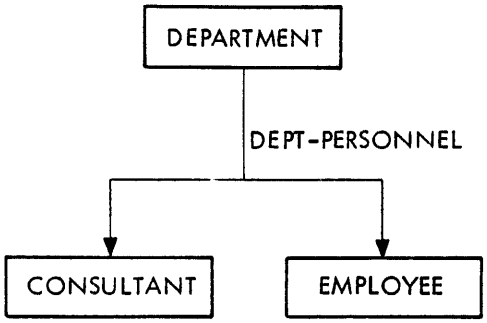


Figure 1. Shorthand Notation for Data Relationships

Though not shown in Figure 1, sets with two or more member groups are legal configurations. For example, a group named GROUP-5 could also be defined as a member of SET-D. Or referring to the previous example, the DEPT-PERSONNEL set could have a CONSULTANT group as well as the EMPLOYEE group as a member. This configuration would be depicted as follows:



The data relationships are incorporated in the database by means of set pointers. Every group occurrence has a NEXT pointer for each set in which the group participates (see Figure 2). In addition to the NEXT pointers, occurrences of member groups may have OWNER pointers as illustrated in Figure 3, and both member and owner group occurrences may have PRIOR pointers, as illustrated in Figure 4. Only the NEXT pointers are always inserted in the database, OWNER and PRIOR pointers are user options. Appendix C describes the database that is illustrated in Figures 1 through 4.

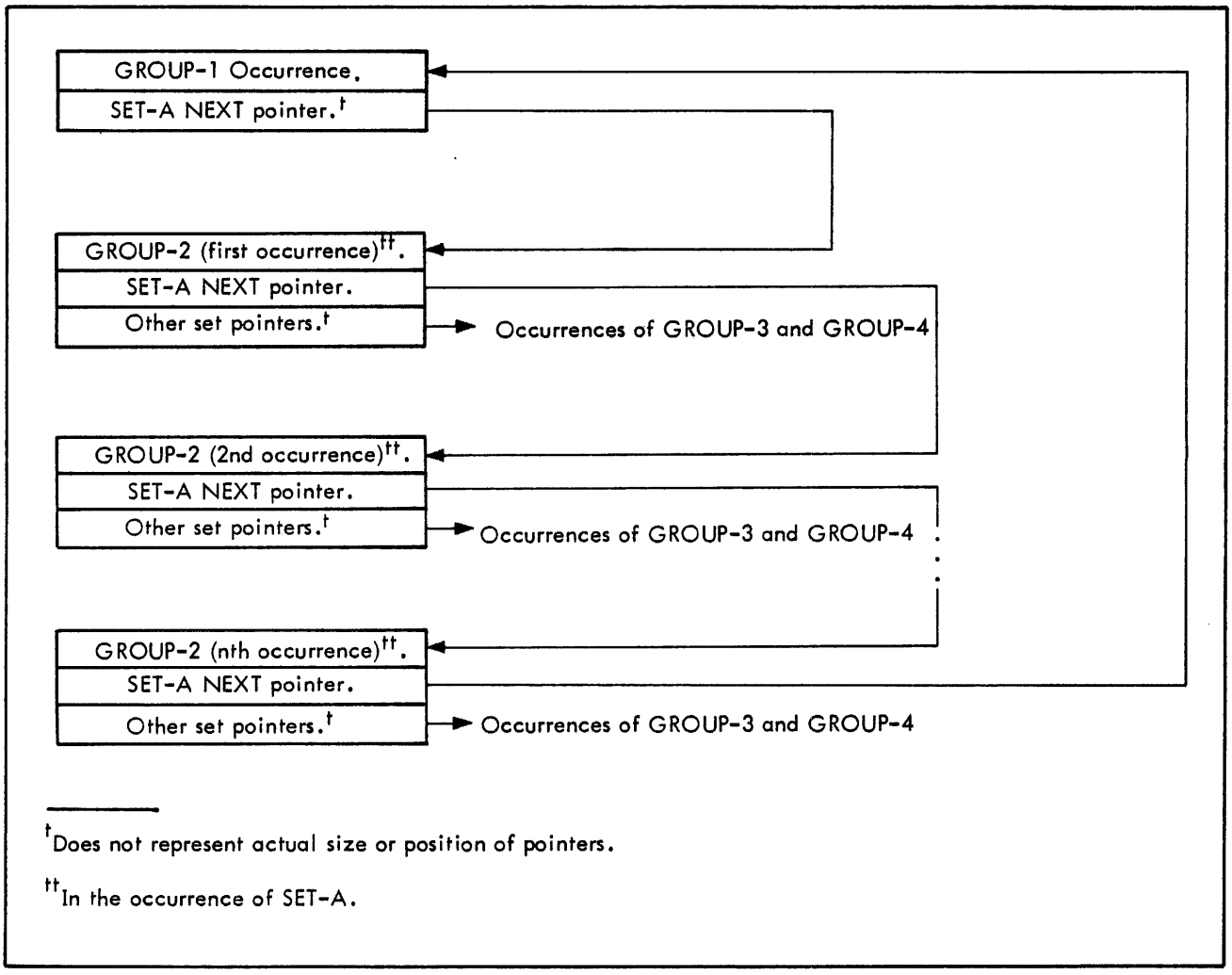


Figure 2. NEXT Pointers in an Occurrence of SET-A

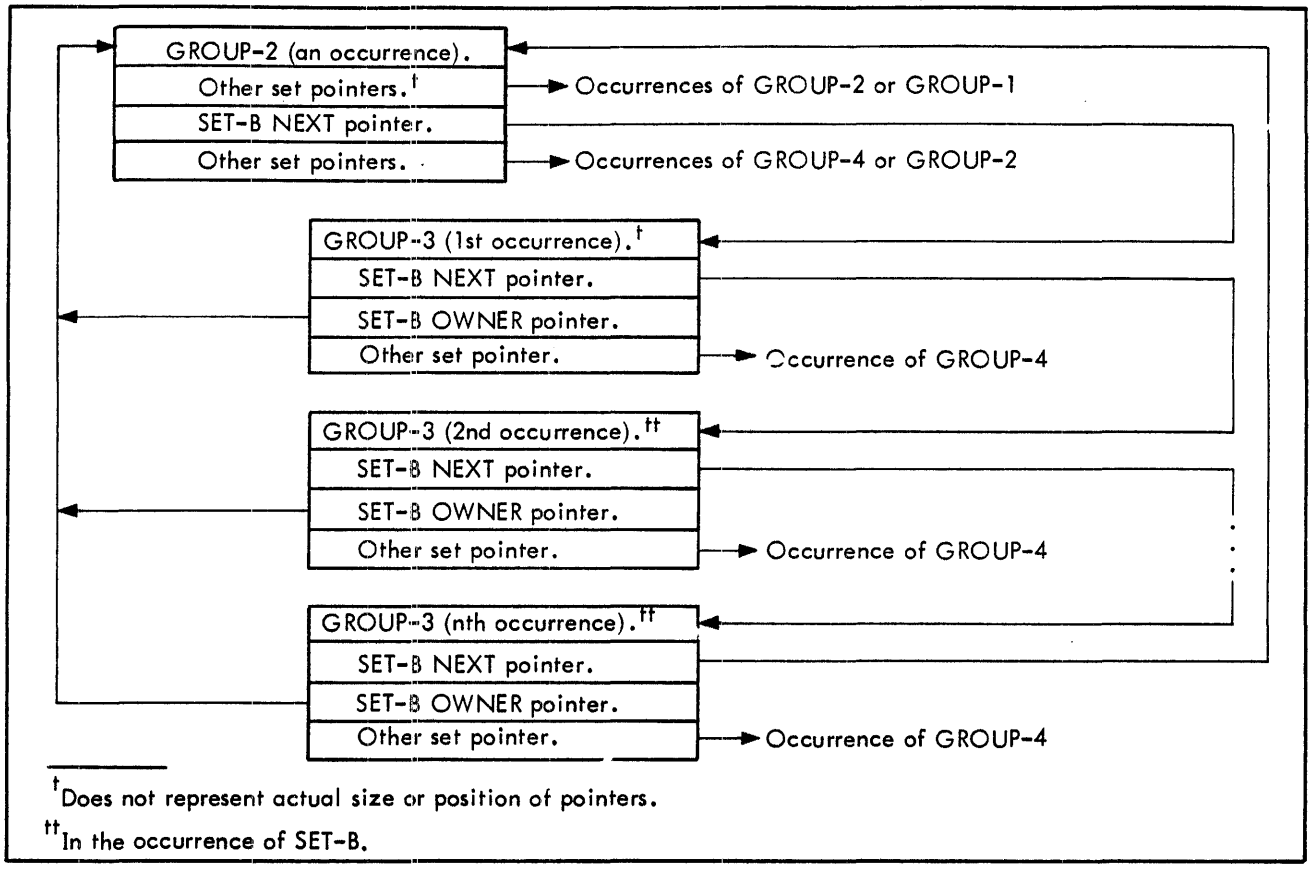


Figure 3. NEXT and OWNER Pointers in an Occurrence of SET-B

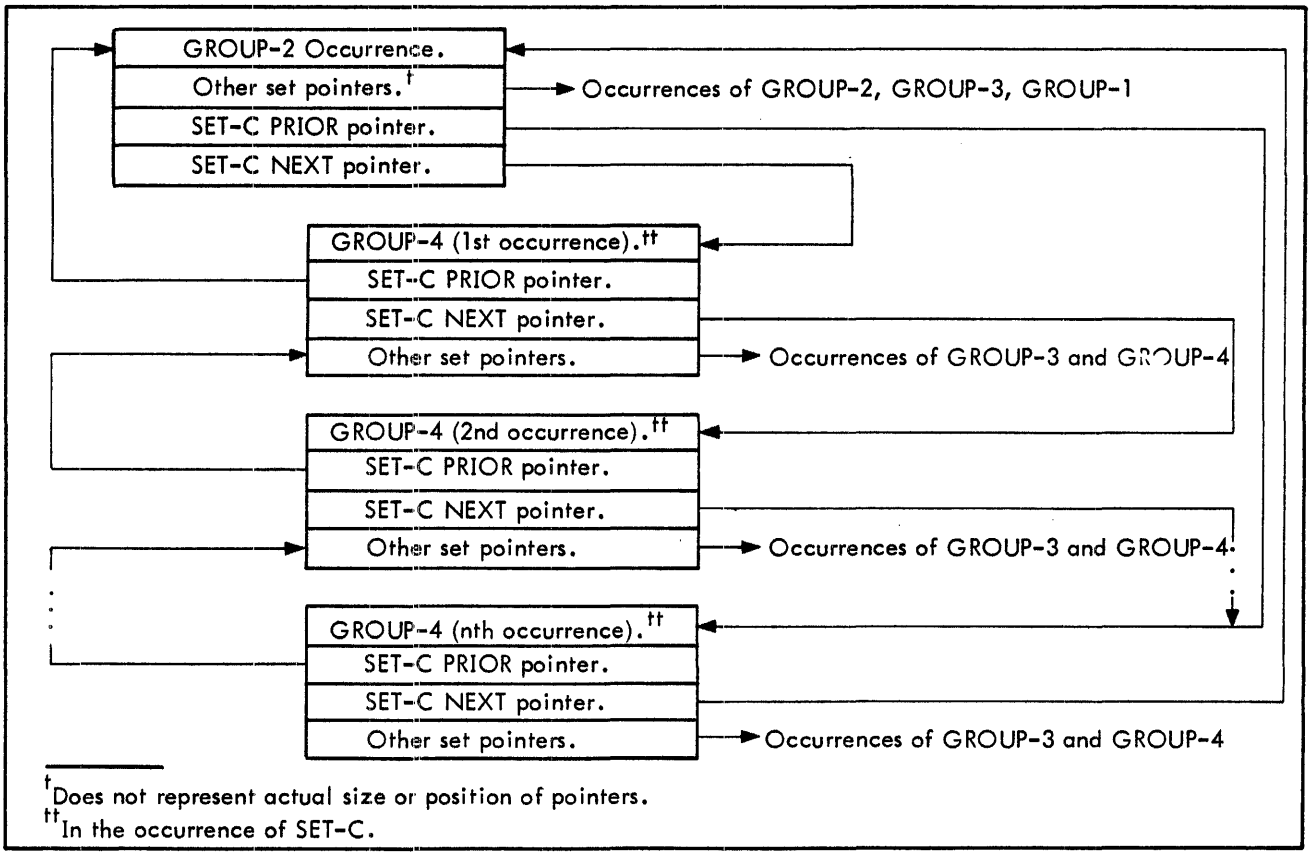


Figure 4. NEXT and PRIOR Pointers in an Occurrence of SET-C

System Functions

The combination of free-standing processors and library routines that comprise EDMS perform five basic categories of system functions:

- Database Definition.
- Database Initialization (null values).
- Data Manipulation (storing, updating, retrieving, etc.).
- Auxiliary Support (maintaining security and integrity, collecting and printing statistics, supplying debugging support to user's programs, etc.).
- Database Restructuring (altering an existing database).

See Figure 5 for a graphic representation of EDMS and Figure 6 for a graphic representation of the restructuring subsystem.

The definition function, centralized in the File Definition Processor (DMSFDP), provides for user specification of database file size, item, group, and set characteristics, and security and integrity requirements. Definition is the required first step in any database activity, and affects the performance of all subsequent functions.

Database initialization prepares the database files for receiving group occurrences. This step is necessary before any actual data values can be added to the database. It creates the complete, maximum-size files, with pages left blank except for control information. This step is performed by a free-standing utility processor, DMSINIT.

Data manipulation is the actual storing, retrieving, and changing of data values. It is performed, in response to user program requests, by the set of library routines referred to collectively as the Data Base Manager (DBM). A working storage area in the user's program, in a format determined by the database definition, is used for communication with the DBM, which performs any necessary file manipulation.

Auxiliary support functions include ensuring database integrity by saving copies of the files, journaling changes, tracing program action, keeping and printing statistics, and other techniques. These features are provided partly by the DBM, and partly by three utility processors, DMSDUMP, DMSLOAD, and DMSSUMS.

Database restructuring provides facilities for altering database size and format. These facilities are provided by two free-standing processors, DMSANLZ and DMSREST.

Database File Structure

The EDMS database exists in random access storage (RAD or disk) as one or more areas, each of which is a file recognizable by the host operating system. EDMS subdivides each area into 512-word page segments. There are three types of pages: data, inventory, and index pages. The number of data pages in each area is specified when the database is defined. If the EDMS inventory facility is selected, one inventory page is added for each 2032 data pages in the area. Pages for the primary index are added if the area is designated for storage of group occurrences in index sequential order. Each area may contain from 1 to $2^{20}-1$ (1,048,575) pages. Pages are numbered consecutively within each area, from 1 to the number defined for the area, plus the number added for inventory and index.

Data Pages

Data pages are used for storing the group occurrences in the area. A data page has a two-word page header and may contain as many as 256 group occurrences and an optional checksum. (See Appendix D, Figure D-1, for an illustration of the data page format.)

The maximum number of group occurrences that can be stored on a data page depends on the size of the occurrences and the number of available line numbers. The size of a group occurrence, which is a collection of item occurrences, control data, and set pointers, is determined by the number and characteristics of the items defined for the

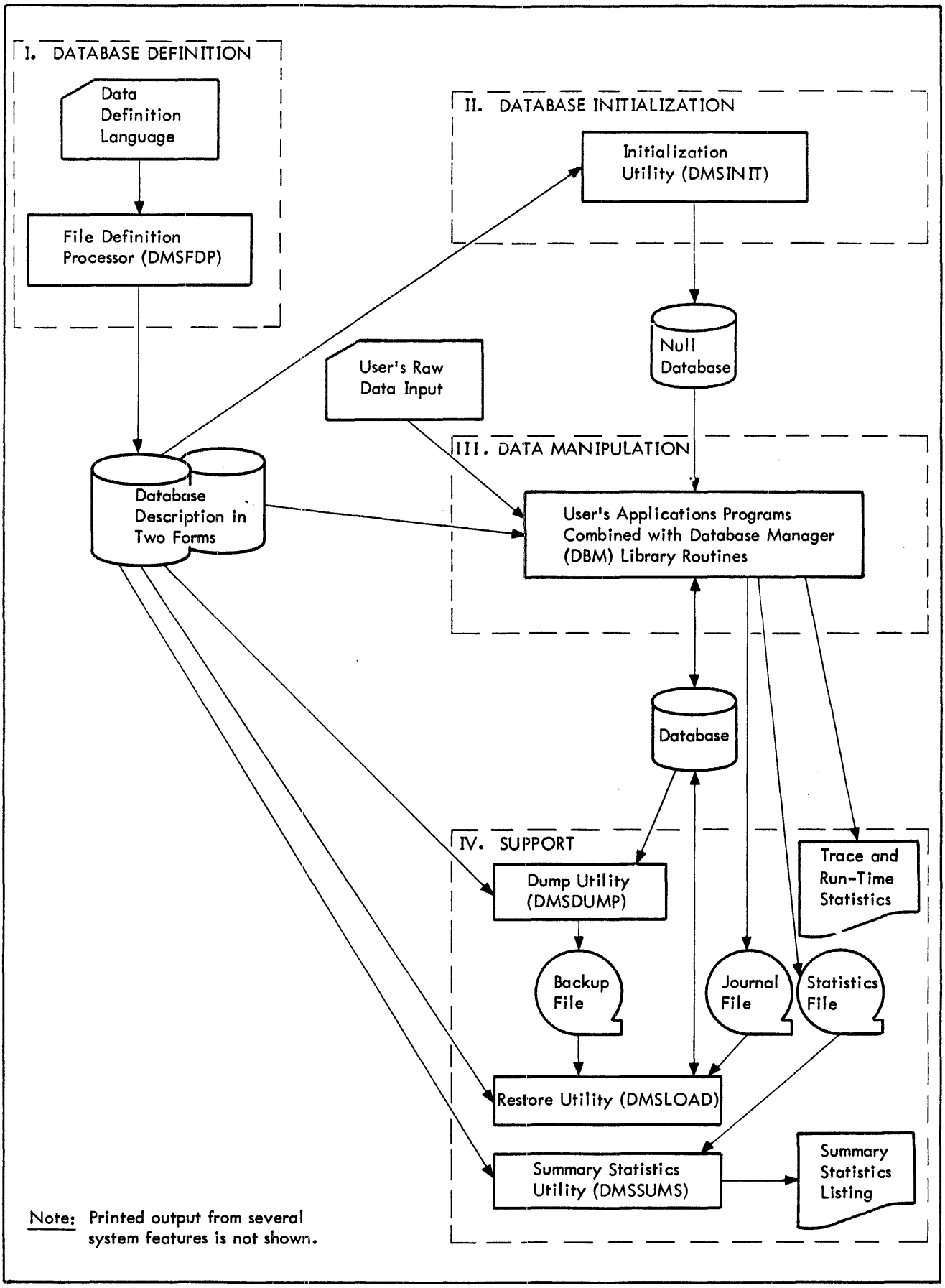


Figure 5. System Overview

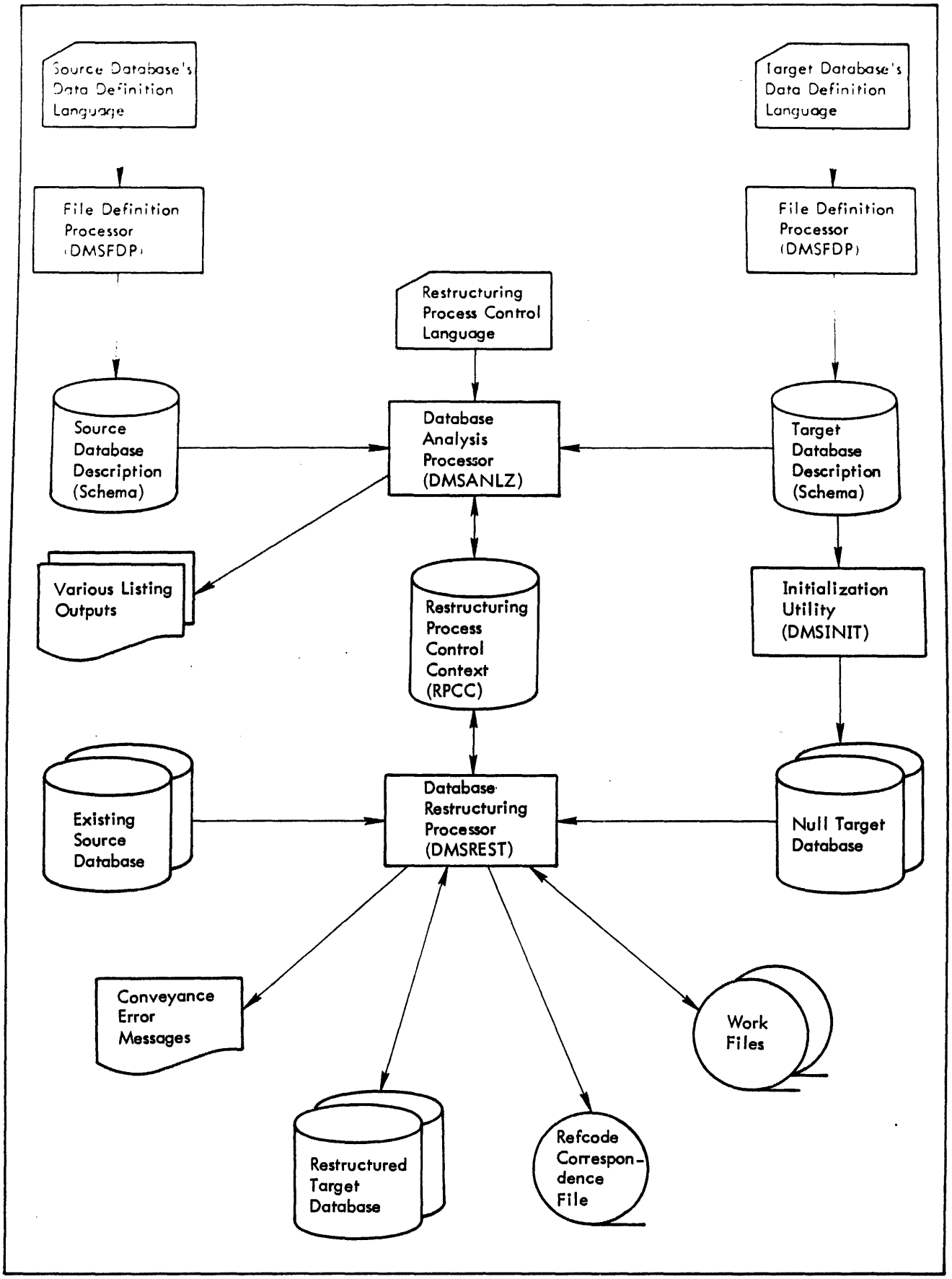


Figure 6. Restructuring Subsystem

group and the number of sets in which it participates. All occurrences of a given group are the same size, but many groups, each with its own size, may be defined for a given database.

The maximum number of available line numbers is determined by the number of pages in the area. When a group occurrence is inserted in the database, it is assigned a line number that is appended to the page number and the area number to form a reference code that uniquely identifies the occurrence. The reference code consists of eight bits of area number and 24 bits shared between page number and line number. The default allocation 24 bits allows representation of the page numbers of all pages in the area, with the remaining bits of the 24 available for line number. For example, if the area contains the maximum number of pages, 20 bits are reserved for page number and only four are available for line number. Similarly, fewer pages allow more bits, up to a maximum of eight, for line number. Thus, in a one-page area or in a 65,535-page area, 16 bits are reserved for page number and eight bits for line number. The user may override the default allocation of bits to allow fewer than the maximum available for line numbers. In a one-area database, set pointers consist of only the 24-bit page-line-number portion of the reference codes. The complete 32-bit codes, including area number, are used for set pointers in databases of two or more areas.

Index Pages

An index page is composed of a three-word page header, a variable number of index entries, and an optional checksum. See Appendix D, Figure D-4, for index page format. The number of pages necessary to contain the indexes is added to the number of data pages specified for an area. Thus, after an area is initialized by the DMSINIT utility, the index pages will follow the data pages of the area. The number of index pages is based on the number of data pages defined to contain group occurrences in index sequential order, and the length of the items defined as the index-key items for the group.

The contents of the index pages are automatically updated by the DBM. As a data page is filled, the highest key value on the data page becomes the index entry in a level-0 index page. When a level-0 index page is filled, the highest key value on that page becomes an index entry on a level-1 index page. The creation of higher level indexes will continue to a maximum of eight levels. The relative position of an entry within an index level corresponds to the relative page number of the page that the entry represents.

Once an index entry is created, it is not removed; i.e., deleting the highest key value on a data page will not change the index for that page.

Inventory Pages

A database area has inventory pages if the user specifies an inventory percentage when he defines the area (see "Area Entries" in the section titled "Schema Generation" in Chapter 3). Each inventory page accommodates space-available counts for 2032 data pages. Figure D-5 in Appendix D shows the inventory page format. The inventory pages, initialized with zero space-available counts by DMSINIT, immediately follow the area's data pages or index pages, if any exist.

The DBM automatically maintains the space-available count for a data page when group occurrences occupy more than the specified percentage of the nonheader words on the page.

Database Restructuring Subsystem

The EDMS Restructuring Subsystem permits the user to change the size or logical structure of an existing database when his requirements change. The Subsystem consists of two free-standing processors, the Database Analysis Processor (DMSANLZ) and the Database Restructuring Processor (DMSREST). DMSANLZ performs an analysis of the user's restructuring requirements and DMSREST uses the results of the analysis to restructure the existing database.

Prior to the availability of a generalized database restructuring facility, a user whose requirements necessitated a change in the database had no recourse except to write special-purpose programs to transfer the data from his existing database into a new database. This process was sometimes prohibitively expensive, both in programming effort and in execution time. Lacking an intimate knowledge of the physical attributes of his database, the user was

required to traverse every set occurrence in his existing database in order to transfer the logical data relationship into the new database. In so doing, he had to access the same data page many times.

The EDMS Restructuring Subsystem minimizes this overhead by accessing the existing (source) database in physical page sequence and outputting the data groups and set linkages to intermediate files. A base or "home" page in the desired (target) database is assigned to each of the data groups and they are then loaded, in one or more physically sequential passes, into the target database. All data groups in the target database are then relinked, in physical page sequence, with the adjusted set pointers.

Subsystem Functions

The requirements analysis function, performed by DMSANLZ, associates the various components (areas, groups, items, and sets) of the source and target databases, checks the legality of all changes, and determines the processes that will be performed by DMSREST and the order in which they will be executed. DMSANLZ conveys this information to DMSREST via the Restructuring Process Control Context (RPCC) file (an EDMS database whose subject is the user's database that is to be restructured). The data in the RPCC file includes descriptions of the various components in the user's database, descriptions (schemas) for the source and target database areas, and a variety of procedural information.

The actual conveyance of the user's data is accomplished by DMSREST on the basis of the information in the RPCC database. Data conveyance consists of a variable number of intermediate processes that may be executed in a single job step or in several successive job steps, at the user's option.

Information Required from the User

All information relating to the structure of the user's database, the physical allocation of space within this database, and any access monitoring and control attributes associated with the user's data are obtained from the schema files for the source and target databases. Thus the user must supply DMSANLZ with two schemas: one for the existing, or source, database, and one for the desired, or target, database. These schemas are referred to in this manual as the source schema and the target schema, respectively.

Any information required to access schema files and database area files must be supplied to DMSANLZ by the user via the Restructuring Process Control Language (RPCL). This information includes schema file names and extract keys, area cipher keys, volume serial numbers for tape files and private disk packs, and account numbers and monitor passwords associated with schemas and database areas. The RPCL may also specify the user's preference with regard to the sequence in which his data groups are to be loaded into the target areas.

The user must provide DMSREST with an existing EDMS database. This may be either in database format (i.e., a random file) or a database dump file created by the Dump Processor (DMSDUMP). The user must also supply DMSREST with an initialized target database. DMSREST is so designed that the source and target databases are not both accessed at the same time.

3. FILE DEFINITION PROCESSOR

The EDMS user defines his database to the File Definition Processor (DMSFDP) in terms of items, groups, sets, and areas. DMSFDP processes the user's definition, stated in a Data Definition Language (DDL), and converts it to a form that is usable by the Database Manager (DBM). The conversion is in two phases. The first phase results in a schema and a listing of error messages, summary information and, optionally, the DDL input.

The schema is established as a file, resident on a random access device. This file contains the names and descriptions of all the items, groups, sets, and areas of the database, and is available for use by the second-phase DMSFDP and the EDMS utilities. Because of its size and complexity, the schema is an inefficient tool that cannot be used by the DBM in directly controlling application program interface with the database. Instead, a subschema, resulting from the second phase of DMSFDP, is used by the DBM as a guide for processing the database.

The second phase of DMSFDP also develops the subschema-specific working storage format that is required for user-program communication with the DBM. Declarations to generate the required formats may be output in files suitable for use in assembling/compiling the user's applications programs, as may listings of the declarations and of the subschema DDL. Figure 7 illustrates DMSFDP outputs and their use in other processes.

Data Definition Language Syntax

The major element of the DDL is the entry. A DDL entry is either a simple entry consisting of one subentry, or a compound entry consisting of two or more subentries. A subentry is composed of one or more clauses and is terminated by a period. The first clause in the first (or only) subentry of an entry identifies the entry, and the first clause in the second, or a succeeding, subentry identifies the subentry. Every clause after the first in a subentry starts with a word, optionally preceded by a semicolon, that identifies the clause. The second and subsequent clauses in a subentry may appear in any order, but the syntactical units within a clause must appear in the specified order.

Clauses consist of words, which include system words and user-generated names, and literals. A word is a string of not more than 30 characters selected from the letters A through Z, the digits 0 through 9, and the hyphen. A word may not begin or end with a hyphen and must have at least one nonnumeric character. Although many system-words having a special meaning in their DDL may also appear as user-generated names, some would result in ambiguity if so used and are reserved for the system. These reserved words are listed below, along with some system-generated names which must not be duplicated by user names.

ALIASES	END	PRIVACY
ALL	FOR	SCHEMA
ARE	GROUP	SET
AREA	INVERT	SET-TABLE
AREA-MASTERS-xx[†]	IS	STATISTICS
AREA-TABLE	KEY	STORAGE
CCB	MEMBER	SYSTEM
COMPONENTS	NAME	THRU
COPY	NUMBER	USING
DUPLICATES	ON	WITHIN

Literals can be numeric or nonnumeric. A numeric literal is a string of characters selected from the digits 0 through 9, the plus sign, the minus sign, the decimal point, and the letter E. Integers, the most commonly used numeric literals in the DDL, are composed of digits only. The number of digits allowed in an integer depends on its use in a clause. Noninteger numeric literals appear only in CHECK clauses (see "Group Entry" in the section titled "Schema Generation", below).

A nonnumeric literal is a string of characters enclosed in a pair of apostrophes. To include an apostrophe in a literal, two apostrophes must be used. The second apostrophe does not become part of the literal. Nonnumeric

[†] x represents any digit.

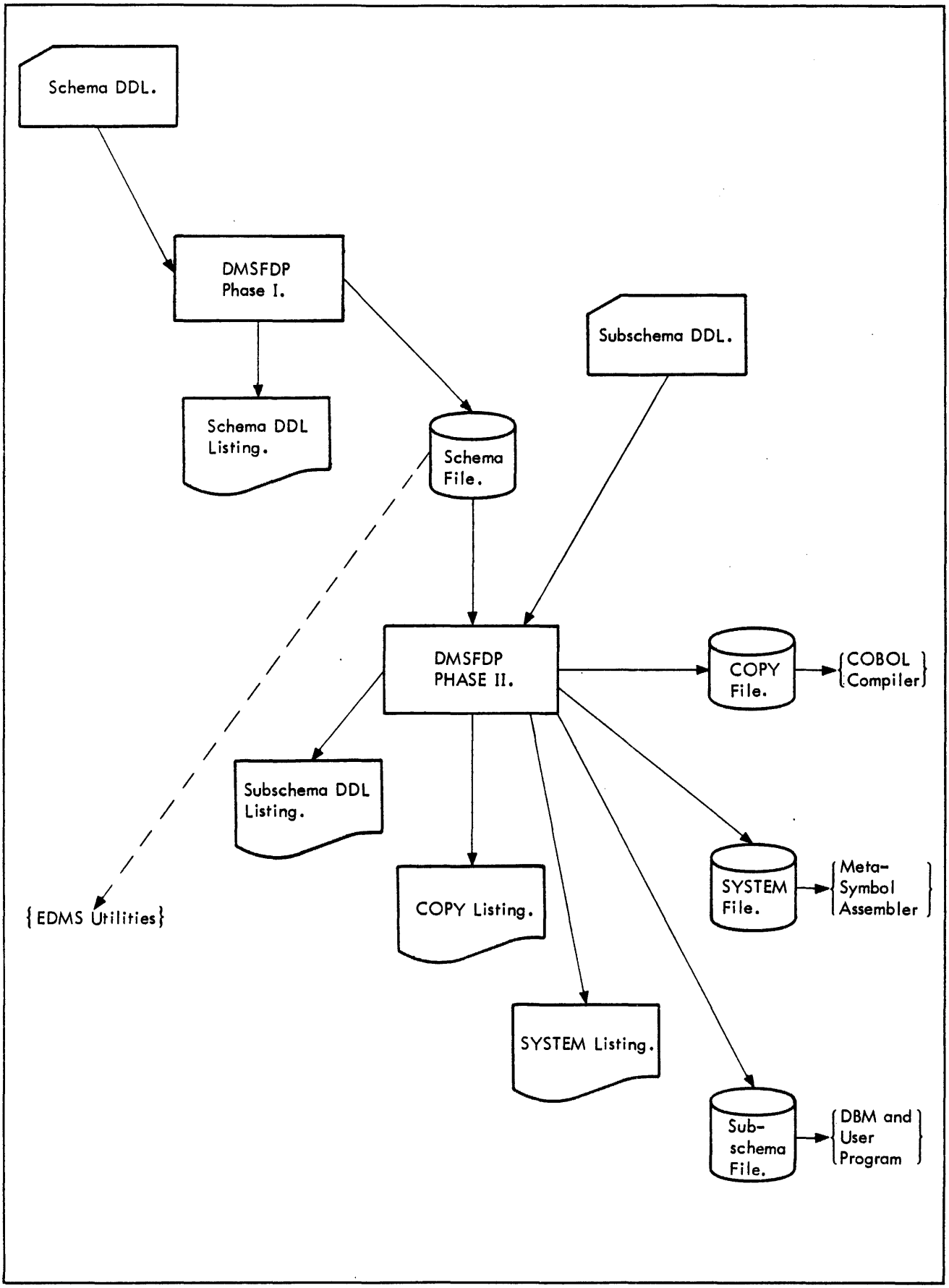


Figure 7. DMSFDP Outputs

literals are used for passwords and privacy locks (see Schema Entry and Subschema Entry, below) and in CHECK clauses. The specific usage determines the allowable size.

The space, the comma, the period, and the semicolon are considered punctuation marks (except in comments and nonnumeric literals) and are used as follows:

1. The space (blank) is a separator, required after words and literals in the absence of any other separator. A space may precede or follow any other separator, and many spaces are the same as one (except in comments and nonnumeric literals).

2. A comma is a separator that is legal only where it is specifically indicated in a language format. The comma, where it is legal, can also serve as a terminator for words and numeric literals. The comma is never required.

3. A semicolon is an optional separator that may be used between clauses but is not needed to indicate the end or beginning of a clause. The semicolon, where it is legal, can also serve as a terminator for words and numeric literals.

4. The period (followed by a space) is required to terminate an entry or subentry.

A comment may be included at any point where a space is legal. Comments are delimited on the left by the contiguous characters /* and on the right by */. A comment may not contain the */ character pair.

The DDL is essentially free-form in terms of length (up to 80 characters) of units of input. The input "unit" is termed a line, though the original input source may be cards, keyboard terminal messages, or any other character-string source. There is no provision in the language for designating a continuation to a new line (card, etc.). An entry or subentry is considered continued until it is terminated by a period, regardless of the number of lines used. However, the end of a line terminates a word or numeric literal.

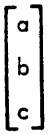
In this manual, the following notation is used to show the DDL entry/subentry format:

1. An underlined word in upper case is required if the part of the format containing it is used.

2. Uppercase words not underlined are optional, but are legal only in the indicated positions.

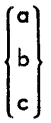
3. Words in lower case represent names or values that are supplied by the user.

4. Brackets indicate that the enclosed part of the format is optional. If two or more language elements are vertically stacked within brackets, none of the elements is required and no more than one may be included. For example,

   a, or b, or c, or none.

5. Braces indicate a required choice. Of the two or more elements vertically stacked within braces, only one may be used, and one is required. For example,

   a, or b, or c.

6. An ellipsis indicates that repetition is allowed. The portion of a format that may be repeated is the total enclosed element whose outermost right bracket or brace immediately precedes the ellipsis. For example,

   [[a b]c]. . . The whole sequence a b c may be repeated.

   [[a b][c]. . .] Only c may be repeated.

Schema Generation

The DMSFDP processes schema DDL and creates a schema file, an EDMS database whose subject is the user's database being defined. The data values in the schema database describe the areas, groups, items, and sets of the user's database. The schema database is described in detail in Appendix A. The schema DDL provides the data input for the schema database as well as information about the schema file itself.

The schema DDL consists of five types of entries.

1. Schema entry – one only.
2. Area entry – one for each area of the database.
3. Group entry – one for each group defined in the database.
4. Set entry – one for each set in the database.
5. End entry – one only.

The schema entry is required and must be the first DDL entry. It is followed by the area entries (at least one), which are followed by the group entries (at least one), which are followed by the set entries (none required). The end entry is the last schema entry. The schema, area, and end entries are simple entries, each consisting of a single subentry. Group entries may be simple or compound. Set entries are always compound, with at least two subentries.

Schema Entry

The schema entry supplies the file name for the schema file and specifies locks and passwords for limiting access to the schema itself and to the user's database.

Format

SCHEMA NAME IS schema-name

[; PRIVACY LOCK FOR EXTRACT IS privacy-lock-1]

[; PRIVACY LOCK FOR ALTER IS privacy-lock-2]

[; PASSWORD IS password-1 [, {RETRIEVE / UPDATE} [KEYS ARE / KEY IS] integer-1[, integer-2]...]...]... .

Usage Rules

1. The SCHEMA clause, which must be the first clause in the entry, specifies the file name for the schema. The specified schema-name must conform to the file naming conventions of the host operating system as well as to the DDL rules for names.
2. The PRIVACY LOCK clauses specify the locks to be used to prevent unauthorized subschema generation using the schema (EXTRACT) and unauthorized modifications (ALTER) of the schema file. (The ALTER lock is not currently used and is provided for use by future enhancements.) The form for privacy-lock-1 and privacy-lock-2 is a nonnumeric literal of up to eight characters. If fewer than eight characters are specified, blanks are added on the right to make an eight-character lock. A key that exactly matches the EXTRACT lock must be supplied in the subschema entry (see "Subschema Generation", below) when a subschema is to be generated.
3. The PASSWORD clause provides information for the DBM to use in controlling access to the user's database. A user's program must supply the DBM with one of the specified passwords to gain access to any database area. The passwords are specified as nonnumeric literals of up to eight characters. Blanks are

added on the right to make eight characters if fewer are specified. Any number of passwords can be specified, within the limits of physical storage space available for the schema file. Access to individual groups and items may be further controlled by the RETRIEVE/UPDATE keys, specified as integers from 1 throug.. 255. A user program is allowed to access the groups and items whose retrieve/update keys match those associated with the password it supplied. (See description of DDL group and item entries, below.) From 0 to 255 retrieve keys and from 0 to 255 update keys may be specified for each password.

Area Entries

Area entries supply (1) the file names by which the database areas are identified for the host operating system and in the user's working storage declarations generated by the FDP; (2) information on the size of the area file; and (3) information on how the file space is to be managed by the DBM.

Format

AREA NAME IS area-name-1 CONTAINS integer-1 PAGES

; NUMBER IS integer-2

[; INVENTORY PERCENT IS integer-3]

[; CHECKSUM IS [NOT] REQUIRED]

[; JOURNAL IS [NOT] REQUIRED]

[; ENCIPHERING IS [NOT] REQUIRED]

[; OVERFLOW RANGE IS PAGE integer-4 THRU PAGE integer-5]

[; FILL PERCENT IS integer-6]

[; LINES PER PAGE [IS / ARE] integer-7].

Usage Rules

1. The AREA NAME clause must be the first in the area entry. Since area-name-1 is subsequently used by the EDMS initialization utility (see Chapter 5) for the file name of the area, the name must conform to the file-naming conventions of the host operating system as well as to the DDL rules for names. The mandatory CONTAINS subclause, which must immediately follow the NAME subclause, specifies the number of data pages required for all occurrences of all groups defined as within the area, including groups defined in invert.subentries (see "Group Entries", below). The EDMS initialization utility calculates the size of the area file by adding (to the number of pages specified) the number of pages, if any, required for inventory and indexes. The number specified by integer-1 must be low enough to ensure that the total area size is not greater than 1,048,575.

2. The required NUMBER clause provides a unique integer identifier for the area. The number specified by integer-2 forms the area-number part of the reference codes for group occurrences in the area. The value specified for integer-2 must be in the range 1 to 64, inclusive, and must not duplicate the number of any other area in the database.

3. The INVENTORY clause indicates that inventory pages are to be included in the area, and specifies the percentage of data words on a page that may be occupied by group occurrences without requiring maintenance of space-available counts. (Data words here means any words not required for header or checksum.) Integer-3 must be in the range 50 to 99, inclusive. For example, INVENTORY PERCENT IS 50 means that space-available counts are to be maintained for all data pages on which more than 255 words (254 if there is a checksum) are occupied by group occurrences. If the inventory clause is not included, no pages will be added to the area file for inventory.

4. The CHECKSUM clause indicates whether or not arithmetic checksums are to be included on the EDMS data pages to provide an error detection capability. If the checksum clause is not included, the data pages will be checksummed, so the clause is needed only if the NOT option is desired. CHECKSUM NOT

is illegal if an ENCIPHERING IS REQUIRED clause (see below) is included. The DBM and the EDMS Utility routines generate and monitor checksums when the data pages are written and read. The user receives an indication if a checksum error is detected.

5. The JOURNAL clause indicates whether or not a journal file is to be maintained when a user program updates the database. (See "Journaling", in Chapter 4.) If the journal clause is not included, no journaling will occur. Specifying JOURNAL NOT, therefore, has the same effect as omitting the clause.

6. The ENCIPHERING clause indicates whether or not the area's data pages and index pages are to be enciphered before being written in the file. Specifying ENCIPHERING IS REQUIRED causes the DBM to use a four-byte key-value supplied by the user's program at run time to modify the words on each page so that they cannot be easily interpreted. To access the data in the area, the user must supply to the DBM or to the EDMS utility routine the same value that was used as a key to encipher the pages. Pages are always checksummed before enciphering, and the checksum is tested after the deciphering. A checksum error indication from the DBM or from an EDMS utility may, therefore, signal either a data error or an improper enciphering key. Specifying ENCIPHERING IS NOT REQUIRED or omitting the enciphering clause indicates that the pages are not to be enciphered.

7. The OVERFLOW clause has meaning and is legal only if a group with location mode of indexed is defined as within the area (see "Group Entries", below). Integer-4 specifies the first, and integer-5 the last,page of a range that is to be reserved exclusively for overflow from the range specified for the indexed group. The overflow pages will be used when a group occurrence that would normally be stored on a page within the indexed group range will not fit on that page. (See "Adding Occurrences" in Chapter 4.) Integer-4 must be one or greater and integer-5 must be less than or equal to the total number of data pages specified by integer-1 in the CONTAINS subclause.

8. The FILL PERCENT clause is also applicable and legal only if the area is to contain indexed group occurrences. The percent specified by integer-6 controls the number of words on a page within the page-range of the indexed group that will be used for storing group occurrences when the area is first created. Integer-6 may be any integer from 1 through 100. (Specifying 100 is the same as not specifying fill percent.) The percent specified by integer-6 is applied to 510 (or 509, if checksum is specified) to determine the maximum number of words to be used while the area is open in create mode. (See "Begin Processing", in Chapter 4, for an explanation of open in create mode.) It is the user's responsibility to select a reasonable percentage figure based on the size of his group occurrences and the relative number of occurrences he will store during create mode.

9. The LINES clause allows the user to decrease the default value for the maximum number of group occurrences that may be contained in any one page in the area. The default value for the number of lines per page is a function of the number of data pages in the area, as follows:

Number of Data Pages in Area	Default Lines Per Page
1 to 65,535	255
65,536 to 131,071	127
131,072 to 262,143	63
262,144 to 524,287	31
524,288 to 1,048,575	15

Legal values for integer-7 are 15, 31, 63, 127 and 255. The value of integer-7 may not exceed the default lines per page for the area.

Group Entries

Group entries specify the size, form and order of appearance of item values within group occurrences, the method for locating occurrences, the privacy locks that are to control access to the occurrences, and which items, if any, are to serve as secondary indexes. Corollary groups or subgroups, used to manipulate secondary indexes, are defined to designate items as secondary indexes.

A group entry consists of a group subentry, followed by item subentries for all items in the group, followed by invert subentries for all of the corollary groups that control secondary indexes for the main group.

Group Entry Skeleton

Group subentry

First item subentry

:

Last item subentry

First invert subentry

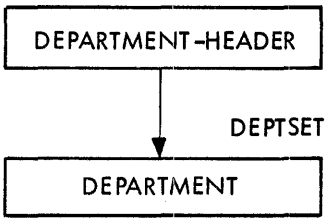
:

Last invert subentry

The required group subentry identifies the group entry. Item subentries and invert subentries are optional, but a group entry with invert subentries must have corresponding item subentries. Other considerations usually necessitate item subentries. Indexed and calc location modes require item values to determine storage and retrieval algorithms; item values are used to determine linking order for member group occurrences in sorted sets; and finally, only items have actual data values; therefore, occurrences of groups with no items are null occurrences, useful only for linking other group occurrences. An itemless group might be useful on two occasions: (1) to serve as the owner of a set that links all group occurrences of a single type, and (2) to serve as a member of two sets and establish connections between specific occurrences of independent groups.

Example 1

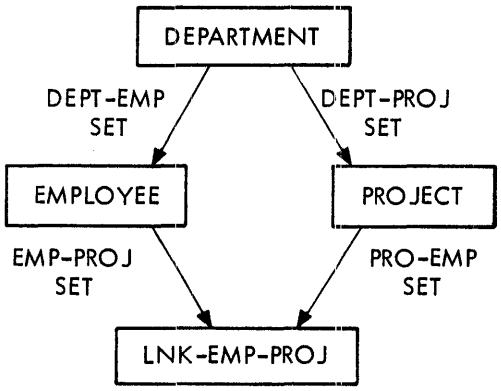
An application needs to access department information in order by department. A simple way to provide for this is to define a set whose sole purpose is to link department group occurrences.



The group named DEPARTMENT-HEADER would not need to have any items (if its location mode were DIRECT, see below), as all the data would be carried in occurrences of the group named DEPARTMENT, which could be accessed through the set named DEPTSET.

Example 2

A department responsible for many projects and with many employees must process project information and employee information and determine which employees are assigned to which project.



Since occurrences of the LNK-EMP-PROJ group serve only to link specific occurrences of EMPLOYEE to specific occurrences of PROJECT, this group does not require any item subentries (assuming its location mode is via one of the sets).

Group Subentry

Group subentries specify the name of the group, the criteria for identifying a specific group occurrence, the guidelines for placing the group in physical storage, and the privacy controls for the group.

Format

GROUP NAME IS group-name-1

; WITHIN area-name-1 [,RANGE IS PAGE integer-1 THRU PAGE integer-2]

; LOCATION MODE IS { DIRECT [,STORAGE IS set-name-1 SET] | INDEXED USING data-item-name-1 [, data-item-name-2]... | CALC USING data-item-name-3 [, data-item-name-4]... DUPLICATES ARE [NOT] ALLOWED | VIA set-name-2 SET [, STORAGE IS set-name-3 SET] }

; NUMBER IS integer-3

[; PRIVACY LOCK FOR RETRIEVE IS integer-4]

[; PRIVACY LOCK FOR UPDATE IS integer-5]

[; STATISTICS ARE [NOT] REQUIRED].

Usage Rules

1. The GROUP NAME clause is required as the first clause in the subentry. The specified name identifies the group for reference in subsequent set entries, subschema selection entries, and in working storage declarations generated by DMSFDP. The name is used as specified for COBOL declarations but may be modified for Meta-Symbol declarations. (See "Subschema Entry" under "Subschema Generation", below.) Group-name-1 must conform to the DDL rules for names and must not be the same as the name specified for any other group or for any item or set in the database.

2. The WITHIN clause specifies the area in which all occurrences of the group are to be stored, with area-name-1 the name of an area defined for the database (see "Area Entries", above). The RANGE subclause specifies the range of pages ($1 \leq$ integer-1 $\leq$ integer-2) in the area on which group occurrences will be stored. The pages are not reserved exclusively for the group, but there are some restrictions on overlapping page ranges if a group with indexed location mode is defined as within the area. No group's range may overlap that specified for OVERFLOW (see "Area Entries", above), and only a limited selection of other groups may be ranged with the indexed group. Specifically, a group's range may coincide with that of the indexed group only if its storage owner may be legally ranged with the indexed group. The storage owner (i.e., the owner of the set specified in the STORAGE clause or the owner of the via set if there is no STORAGE clause) may be the indexed group itself, or it may be a group whose storage owner is the indexed group, etc., down as many levels as desired. The range of a group whose location mode is calc may not overlap the range of an indexed group. Any range that overlaps the range of an indexed group must exactly coincide with it. If RANGE is specified, integer-1 must be greater than or equal to 1 and less than or equal to integer-2, and integer-2 must be less than or equal to the number of data pages specified for the area. If RANGE is not specified, the range used is 1 through the highest numbered data page in the area.

3. The LOCATION MODE clause specifies the most important group characteristic. The location mode determines how the DBM selects physical locations for group occurrences and the primary means by which the user identifies a specific occurrence to the DBM for retrieval and set-linking purposes. It also affects the types of set linkages that are legal for the group. There are four location modes available: direct, indexed, calc, and via set.

   DIRECT – The user identifies a specific group occurrence to the DBM by supplying the reference code that is returned by the DBM when the occurrence is stored. The location selected by the DBM for storing an occurrence depends on whether there is a STORAGE set specified in the definition. If a STORAGE set is specified, a group occurrence will be stored physically near its associated owner occurrence. If a STORAGE set is not specified, the user must supply the area number and may supply a base page number in his working storage for the DBM to use in selecting a physical location. (See "Adding Occurrences" in Chapter 4.) If STORAGE is specified, the set owner must be defined as within the area identified by area-name-1. The group's inclusion in the set must not be manual (see "Member Subentry", below).

INDEXED – Indexed group occurrences are stored in sequential order of increasing key values. A key value is formed by the catenation of the values of the items identified by data-item-name-1, data-item-name-2, etc. From one to seven items may be specified. The number of items used should be sufficient to provide a unique key value for each occurrence of the group, as duplicate key values are not allowed. The highest key value stored on a page is also stored on an index page as the key for the data page. Group occurrences may then be retrieved either individually by means of specific key values, or sequentially in either direction. Not more than one group with indexed location mode may be defined for any given area.

CALC – User supplies, in working storage, the control item values of the specific group occurrence to be retrieved. Group occurrences are stored on or near a base page whose page number is determined through a hashing of the values of the control items identified by data-item-name-3, data-item-name-4, etc. From one to seven control items may be identified, all to be defined in item subentries in the group entry. The DUPLICATES phrase is required in the calc specification. If duplicates are not allowed, a data-dependent error return will be made to a user's program that attempts to store a group occurrence whose combined control item values duplicate those of an existing group occurrence. If duplicates are allowed, more than one group occurrence may have a specific control-item value combination and the user will have to make more than one retrieval request to obtain all group occurrences with that value.

VIA SET – Each occurrence of the group, which must be defined as an automatic member of the set identified by set-name-2, is stored physically near the owner occurrence with which it is associated. However, if a range is specified for the group, the occurrences will be stored within that range regardless of the location of the selected owner occurrences. The set-name-3 set of the optional STORAGE clause replaces set-name-1 set for positioning of occurrences, but it does not override a RANGE specification. The primary means of identifying a specific occurrence of the group to the DBM for retrieval is by relating it to a specific occurrence of the set identified by set-name-2. If STORAGE is specified, the set owner must be defined as within the area, and the group's inclusion in the set must not be manual.

4. The mandatory NUMBER clause assigns a unique integer identifier to the group. All occurrences of the group will contain this number, which will also be part of the working storage identifier used to store the reference code of the most recently accessed occurrence of this group (see the description of "Current-of-Type" under "Adding Occurrences" in Chapter 4). The value of integer-3 may range from 1 to 999, but must not duplicate the value assigned to any other group defined for the database.

5. The PRIVACY LOCK clauses supply lock values (integers 1 to 255) that DBM and the dump utility use to determine if a user has authority to retrieve or update the group occurrences. If locks are specified, group (or item) occurrences cannot be retrieved or updated unless a key that matches the lock is associated with the password supplied to the DBM in the user program's working storage or as input to the dump utility. The value of integer-4 and integer-5 must, therefore, match appropriate keys specified in a PASSWORD clause in the schema entry.

6. The STATISTICS clause indicates whether or not the DBM is to keep summary-type statistics when group occurrences are stored, retrieved, or deleted. If the clause indicates that STATISTICS ARE REQUIRED, the DBM will collect the statistics automatically during user program operation, though the user must assign a file (see DBM "Operational Interface" in Chapter 4) for storing the statistics. If NOT is specified or if the clause is omitted, no summary statistics will be kept on the group.

Item Subentries

Item subentries specify the characteristics of the items in the group. All of the item subentries for a group together provide an image of the data portion of the group occurrences in the database. The item values exist in the group occurrences in the exact order in which the item subentries occur, with no intervening slack bytes. For this reason, the order of the item subentries can affect the efficiency of subsequent accesses of the defined database. For greatest efficiency, the item subentries should be arranged in an order that results in binary and floating-point (long and short) item values beginning on word boundaries.

Format

data-item-name-1

[; {PICTURE / PIC} IS character-string]

; TYPE IS { BINARY | FLOATING {SHORT / LONG} | PACKED DECIMAL[, integer-1] | CHARACTER[, integer-2] }

[; OCCURS integer-3 TIMES]

[; PRIVACY LOCK FOR RETRIEVE IS integer-4]

[; PRIVACY LOCK FOR UPDATE IS integer-5]

[; CHECK IS {PICTURE / RANGE OF literal-1 THRU literal-2}]...

Usage Rules

1. Data-item-name-1 must appear first in the item subentry, must conform to the DDL rules for names, and must not be the same as the name specified for another item in the group or for any set or group defined for the database. The specified data-item-name is used in the working storage declarations that are generated for use in COBOL and Meta-Symbol applications programs. (In the COBOL definition the name appears as specified, but it may be modified for Meta-Symbol usage; see "Subschema Generation", below.)

2. The PICTURE clause may be used to indicate the form of the values of certain types of items. The picture is included in the COBOL working storage declarations and may be used by the DBM to perform validity checks on input data values (see CHECK clause, below). Characters in the picture character-string represent characters and character positions in data values. The picture-character-string characters have the following meaning.

   A – letter or space

   X – any character

   9 – digit

   V – assumed decimal point

   P – assumed scaling position

   S – sign (+ or -) – must be leftmost character if used.

   To indicate a number of characters, the representative character (except S) may either be repeated or followed by an integer enclosed in parentheses. For example, AA and A(2) both signify two letters. The maximum number of characters in the picture character string is 30. The maximum item size depends on a combination of the picture information and the specified item type. The PICTURE clause is required if the TYPE clause (below) is not included, and is illegal for certain values of TYPE (see Table 1).

3. The TYPE clause is used in conjunction with the PICTURE clause to determine (1) the database representation of the item values and (2) the method DBM uses to process the values. The allowable item size depends on the TYPE-PICTURE combination. The TYPE clause may specify item size if TYPE is PACKED DECIMAL (integer-1) or CHARACTER (integer-2). If a size is specified, it must be the same as the size implied by the picture clause. The size is required if the PICTURE clause is omitted. Table 1 shows PICTURE-TYPE relationships, the EDMS interpretation of each combination, and the allowable item sizes for each.

Table 1. PICTURE-TYPE Correspondences

Type	Picture	DMS Interpretation	Size
Binary	Illegal	Binary	Fixed — one word.
Floating Long	Illegal	Double Precision Floating-Point	Fixed — two words.
Floating Short	Illegal	Single Precision Floating-Point	Fixed — one word.
Packed Decimal	9's, P's, S, and V	Packed Decimal	Variable — maximum 31 digits (16 bytes).
Character or not specified	9's, P's, S, and V	Signed Numeric	Variable — maximum 31 digits (31 bytes) (only 9's counted).
Character or not specified	9's, P's, and V (no S)	Numeric	Variable — maximum 31 digits (31 bytes) (only 9's counted).
Character or not specified	A's	Alphabetic	Variable — maximum 255 characters.
Character or not specified	X's, or A's, X's, and 9's	Alphanumeric	Variable — maximum 255 characters.

4. The OCCURS clause indicates the number of times an item value is repeated in a group occurrence. The size of the group occurrences will be made large enough to accommodate an item that is integer-3 times the size of the specified item. EDMS will treat the total as one large item. The OCCURS clause must not be included if the item is a control item for a calc or indexed group, if the item is a sort key for a set (see "Set Entries", below), or if the item is a secondary index item (see "Invert Subentries", below).

5. The PRIVACY LOCK clauses have the same effect as those in the group subentry except that the locks are for the item values only. Authority to access a group does not imply authority to access all items if any item has a privacy lock.

6. The CHECK clause indicates that the DBM is to validity-check values supplied for the item when a group occurrence is stored or modified. Refer to Appendix G for a discussion of data validation by the DBM. If PICTURE is specified, an attempt to store an item value that does not agree with the item's PICTURE clause will result in an error return from the DBM. PICTURE is not allowed in a CHECK clause if there is no PICTURE clause.

   If RANGE is specified, an attempt to store an item value that is less than literal-1 or greater than literal-2 will result in an error return from the DBM. The values specified by literal-1 and literal-2 may be equal and must be compatible with the item's size and form, as determined by the PICTURE-TYPE combination. The RANGE option is not legal if the item size amounts to more than four words of computer storage. Literal-1 and literal-2 may be numeric or nonnumeric literals, depending on the item.

   A numeric literal is a string of characters selected from the digits 0 through 9, the plus sign, the minus sign, the decimal point, and the letter E. Rules for the formation of numeric literals are

   a. The literal must contain at least one digit.

   b. The literal may contain at most two sign characters. A sign character is legal as the leftmost character of the literal and immediately to the right of the letter E. If either sign character is omitted, a positive value is implied.

   c. The literal must not contain more than one decimal point, which must be to the left of the letter E. If no E is included, the decimal point may appear anywhere in the literal except as the rightmost character. The number of digits to the left of the E must not be greater than 31 or less than 1.

A nonnumeric literal is a string of any characters (up to 16) enclosed in apostrophes. If the value is to contain an apostrophe, two apostrophes must be included.

Invert Subentries

Invert subentries identify the items in the group that are to serve as secondary indexes, providing an alternative technique of identifying specific group occurrences for retrieval. (The primary technique is determined by the group's location mode.) A secondary-index-item value (supplied by the user in his working storage) can be used in the retrieval of the group occurrences in which that value exists.

The secondary index capability is implemented in EDMS by means of a corollary group, called an invert group. An invert group, which has some of the characteristics of a regular calc group, must be defined for each item that is to be a secondary index. Each occurrence of an item identified as a secondary index item causes the item value to be stored in an occurrence of the invert group as well as in the occurrence of the main group in which the item is defined. The occurrence of the invert group consists of the value of the secondary index item and the reference code of the main-group occurrence that contains the value, plus control information and set pointers.

The first invert subentry in a group entry follows the last item entry for the group.

Format

INVERT ON data-item-name-1

; NUMBER IS integer-1

; WITHIN area-name-1 [RANGE IS PAGE integer-1 THRU PAGE integer-2]

; DUPLICATES ARE [NOT] ALLOWED.

Usage Rules

1. The INVERT clause must appear first in the subentry, and data-item-name-1 must be the name of an item defined in an item subentry that does not contain an OCCURS clause.

2. The NUMBER clause provides the unique integer group identifier (see "Group Subentry", above) for the corollary invert group. The value of integer-1 must be in the range from 1 to 999 and must not be the same as the integer specified in the NUMBER clause of any other group defined for the database.

3. The WITHIN clause identifies the area in which occurrences of the invert group are to be stored (see "Area Entries", above). Because the invert group occurrences need not be stored in the same area as the occurrences of the associated main group, the area name in the invert subentry may either be the same or different from that specified in the group subentry. The RANGE subclause specifies the pages within the area on which the group occurrences are to be stored and must be included if a group with indexed location mode is defined as within the specified area. Integer-1 must be greater than or equal to 1 and less than or equal to integer-2. Integer-2 must be less than or equal to the integer that specified the number of data pages in the area (see "Area Entries", above). If there is an indexed group in the area, the range indicated by integer-1 and integer-2 must not overlap its range. Nor may the invert group range overlap the OVERFLOW range (if one was specified).

4. The required DUPLICATES clause specifies whether or not two or more main group occurrences with the same secondary-index item value will be allowed. If DUPLICATES ARE NOT ALLOWED, a user program's attempt to store a group occurrence that would cause a duplicate invert group occurrence will receive an error return from the DBM. If DUPLICATES ARE ALLOWED, more than one retrieval request may be needed to retrieve all group occurrences with a specific secondary-index item value.

Set Entries

The set entries define all the user-specified relationships among group occurrences by indicating which groups are to participate in which sets, what set pointers are to be included in the group occurrences, what is to determine which owner group occurrence a particular member group occurrence is to be associated with, and how the member occurrences are to be associated with each other.

Set Entry Skeleton

Set Subentry

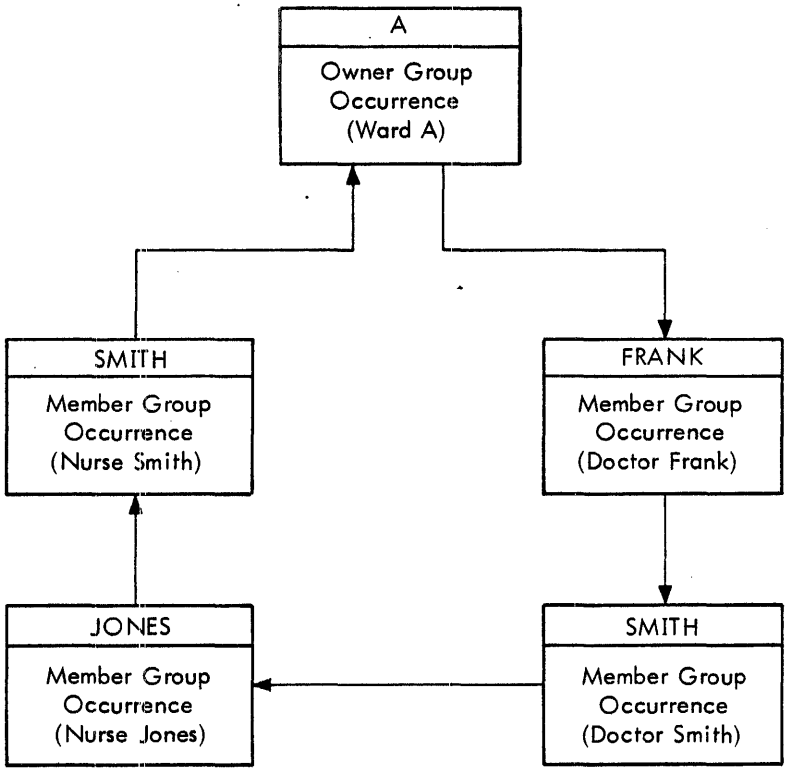
Member subentry

[Member subentry] . . .

Set Subentry

A set subentry provides the name by which the set is referenced in other DDL entries (e.g., in group entries of groups whose location mode is via set, and in subschema set entries), and in DMSFDP-generated working storage declarations; name the group type that is to be the owner of the set; and specify the mode of linking member group occurrences to each other and to the owner occurrence.

A set occurrence is defined as one occurrence of the owner group and a collection of associated occurrences of the group or groups defined as members, as illustrated below for a WARD-ASSIGNMENT set whose owner is a WARD group and whose members are a NURSE group and a DOCTOR group.



(The WARD-ASSIGNMENT set as depicted is in sorted order with group number as major sort key and a NAME item as sort key in both member groups, see below.)

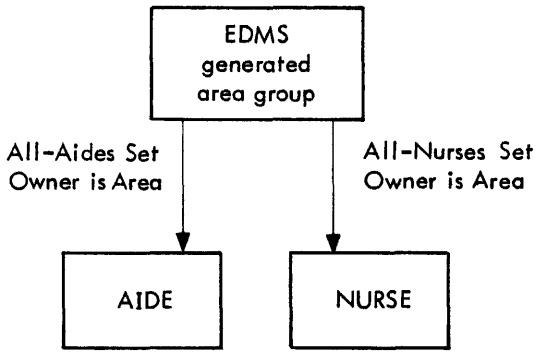
Format

```
SET NAME IS set-name-1
    ; OWNER IS { group-name-1 | AREA area-name-1 }
    ; ORDER IS { SORTED [WITH GROUP-NO AS {MAJOR | MINOR}] | FIRST | LAST | NEXT | PRIOR }
    [; LINKED TO PRIOR]
    [; STATISTICS ARE[NOT] REQUIRED].
```

Usage Rules

1. The SET NAME clause must be the first clause in the subentry. Set-name-1 must conform to the DDL rules for names and must not be the same as the name used for any item or group, or for any other set defined for the database.

2. The OWNER clause identifies the group that is to participate in the set as owner. Group-name-1 is the name specified in the group subentry that defined the group. If AREA is specified, the DMSFDP will generate a group definition for a special group to serve as owner. A single occurrence of this group will be maintained by the DBM, at page 1, line 1, of each area, to serve as the owner occurrence for every set for which the area is owner. A set whose owner is area will, therefore, have only one set occurrence, which will consist of the one area-group occurrence plus all the occurrences of the groups defined as members of that set. The special EDMS-defined area group may be owner of many sets as illustrated in the data structure diagram shown below, where the OWNER IS AREA feature is used to link all the occurrences of the NURSE group (e.g., for all nurses employed at a hospital) to each other, and to link all the occurrences of the AIDE group to each other.

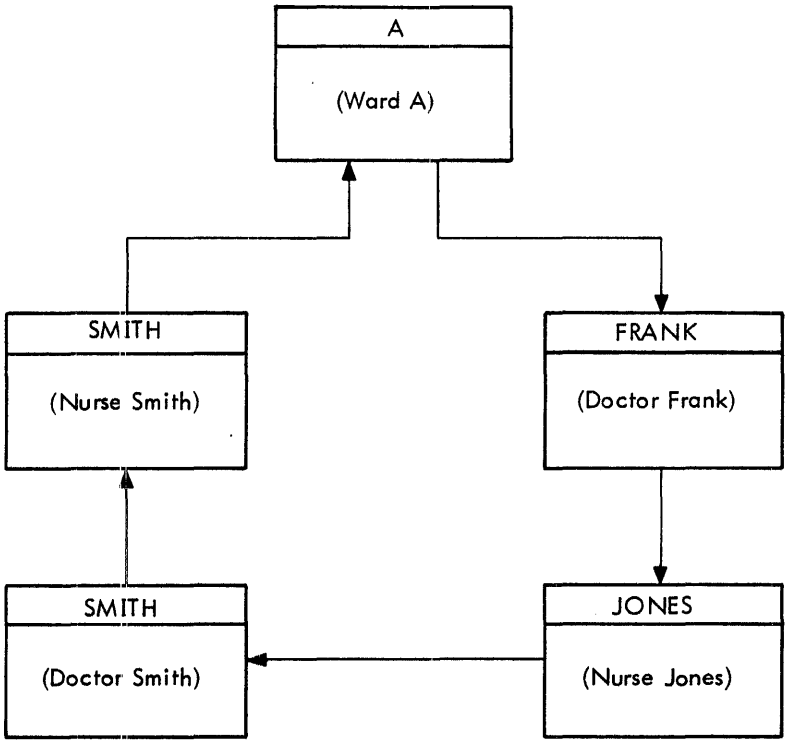


The EDMS-generated area-group occurrence has no data values and is not accessible as a group to the user. It serves only to link occurrences of a member-group to other occurrences of the same member group.

3. The ORDER clause specifies the manner in which DBM is to generate and modify set pointers so that they will link a member occurrence into a set occurrence. It determines if the owner occurrence or a member

occurrence is to be modified to point to the newly linked occurrence and, if a member occurrence, which one. Since set order is applied after the proper owner occurrence has been selected (see "Member Subentry", below), it refers only to logical sequence within a set occurrence. Five modes of pointer maintenance are possible: sorted, first, last, next, and prior.

SORTED -- The DBM links a new member occurrence to other member occurrences according to the values of the data items defined as KEYs in the member subentries. If WITH GROUP-NO is specified, the unique numbers included in the occurrences of the member groups (see NUMBER clause in "Group Subentires", above) will be considered in selecting a set position for a new member occurrence. GROUP-NO is legal only if more than one group type is designated as a member of the set. MAJOR or MINOR defines the role of the GROUP-NO in the order of the set occurrences. The WARD-ASSIGNMENT SET occurrence depicted above is an example of a set sorted with group-no as major (assuming the group subentries specified NUMBER IS 100 for the DOCTOR group and 200 for the NURSE, and both groups had a NAME item designated as an ascending key in a member subentry, see below). If GROUP-NO AS MINOR was specified, the occurrence would appear as follows:



FIRST – The DBM creates LIFO-ordered set occurrences by inserting a new member occurrence as the first occurrence following the owner occurrence. The NEXT pointer for the set in the occurrence of the group designated as owner will point to the most recently linked member occurrence.

LAST – The DBM creates FIFO-ordered set occurrences by inserting a new member occurrence immediately preceding the owner occurrence. This order implicitly defines a prior pointer for the owner occurrence.

NEXT – A new member occurrence is inserted immediately following the occurrence identified as current of the set. This order requires that the user establish a position in a set occurrence (by storing or retrieving the group occurrence to which the new occurrence is to be linked) before linking the new occurrence.

PRIOR – causes a new member occurrence to be inserted immediately before the occurrence identified as current of the set. This order also requires that the user establish a current position in a set occurrence, as well as implicitly defining prior pointers for the owner and member occurrences.

4. The LINKED TO PRIOR clause defines the optional backward pointers for the set's owner and member groups, so that each occurrence of the owner or a member will point to the preceding occurrence.

5. The STATISTICS clause indicates that the DBM is to maintain statistics for the set. If the clause is omitted or if NOT is specified, statistics will not be collected.

Member Subentries

Member subentries identify the groups that are to be members of the set and specify all the controls that are to apply when a new occurrence is stored or whenever a member occurrence is linked into a set occurrence. These controls are (1) the technique for selecting the owner occurrence that the member is to be linked to, (2) whether or not pointers to the owner occurrence are to be included in member occurrences, and (3) the itmes that are to control a member occurrence's logical position in a set for which the specified order is sorted.

Format

```
MEMBER IS group-name-1

    ; INCLUSION IS {[OPTIONAL] AUTOMATIC
                  {          MANUAL

   [; LINKED TO OWNER]

    ; SET OCCURRENCE SELECTION IS THRU

          CURRENT OF SET
          LOCATION MODE OF OWNER[ALIAS FOR data-item-name-1
                                 IS data-item-name-2]...

   [;   [{ASCENDING }[RANGE]KEY IS data-item-name-3]...
        {DESCENDING}

                        DUPLICATES ARE {FIRST      }]
                                       {LAST       }
                                       {NOT ALLOWED}
```

Usage Rules

1. The MEMBER clause, which must be the first clause in the subentry, must specify the name of a group that is defined for the database and not specified in any other member subentry in this set entry.

2. The INCLUSION clause specifies that linking or delinking a member group occurrence from a set occurrence will be AUTOMATIC or MANUAL.

   AUTOMATIC – Member-group occurrences are automatically linked or delinked by the DBM when they are stored or deleted. If OPTIONAL is specified, the occurrences may also be linked and delinked by specific user's calls to the DBM. (See "Linking or Delinking Member Occurrences" in the section titled "Modifying Linkages", Chapter 4.)

   MANUAL – The user will specifically link and delink member group occurrences by calls to the DBM linking/delinking routines. The mode is not legal for the set identified as the VIA set or in a STORAGE clause for the group identified by group-name-1.

3. LINKED TO OWNER defines a set pointer for the member group such that each member occurrence will point to its associated owner occurrence.

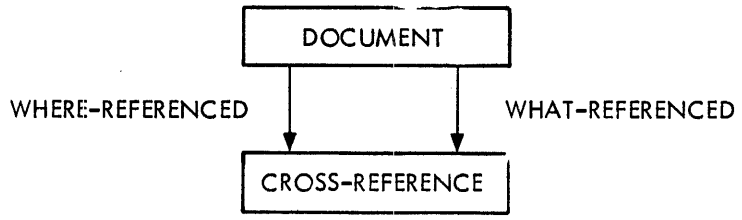
4. SET OCCURRENCE SELECTION specifies the technique to be used to identify the set occurrence into which a specific member occurrence is to be linked.

   CURRENT – Requires that the user establish a set occurrence as current by interacting with the DBM to store or retrieve the owner occurrence or a member occurrence. This is the only mode that is allowed for sets whose order is next or prior or whose owner is AREA.

LOCATION MODE OF OWNER – Indicates that a unique set occurrence is selected by supplying the values required to retrieve the unique owner-group occurrence. If the owner's mode is direct, indexed, or calc, a reference code or specific values for the control item(s) identify a unique occurrence.

If the owner's location mode is via set, there is no way of identifying a unique owner-group occurrence unless the via set is sorted. If the owner's via set is sorted, a unique owner-group occurrence can be identified by specific values for the sort-key items (or approximate values, if RANGE was specified for the key).

ALIASes may be specified to identify additional working storage locations to contain occurrence-selecting values when a group is a member of two or more sets with the same owner, and two or more owner occurrences need to be identified at the same time. For example, the structure shown below could be used to record which documents referenced, or were referenced by which other documents.



Two occurrences of the DOCUMENT group may need to be identified simultaneously to be linked with a CROSS-REFERENCE occurrence. If the location mode of DOCUMENT is calc using DOCUMENT-ID, one occurrence can be identified by supplying the proper value in working storage for DOCUMENT-ID. An ALIAS for DOCUMENT-ID, say DOCUMENT-ID-2, could be defined in the member subentry for CROSS-REFERENCE in one of the sets. This would cause working storage to be available for identifying the other occurrence of DOCUMENT.

Selection through location mode of owner may not be used when the set order is prior or next, or when the owner is AREA.

5. The ASCENDING and DESCENDING subclauses identify the items in the member group that are to be sort-key items for a set in sorted order. Values of the specified items are used (in conjunction with the group number, if WITH GROUP-NO is specified in the set subentry), to establish the logical sequence of member occurrences within a set occurrence. The optional RANGE modifier applies to any sets in which the group identified by group-name-1 participates as owner and in which the set occurrence selection for a member is through location mode of owner. RANGE is not meaningful if group-name-1 does not identify a group whose location mode is via set.

   One ASCENDING or DESCENDING subclause is required if the set is sorted, and up to seven may be specified. Every item specified by data-item-name-3, etc., must be defined as within the member group and defined without OCCURS clauses.

   One, and only one, DUPLICATES subclause must be included if any ASCENDING or DESCENDING subclauses are included. The DUPLICATES specification controls the logical sequence of two or more member occurrences with the same sort-key value, or prohibits duplicate values.

END Entry

The end entry is required after the last set entry. It has the form END.

Subschema Generation

The DMSFDP generates a subschema from a schema as specified in subschema Data Definition Language (DDL). The subschema, which contains the information required for the DBM to identify data values and relationships within the database, may describe a complete database or it may describe only that portion needed for a specific application. It may or may not include the names of the sets, groups, and items it defines.

The contents of the subschema determine the format of a working storage area that the user's program must contain in order to communicate with the DBM. To simplify establishing user's storage to subschema correspondences, the DMSFDP will (optionally) create COBOL COPY files or Meta-Symbol SYSTEM files containing the working storage format definitions that correspond to the subschema it is creating.

The information extracted from a schema to form a subschema may describe either all the components (groups, data-items, sets) of the database, all the components whose occurrences are to be stored in a specified area (or in specified areas), or only selected components. If a subschema is not to describe a complete database, certain rules must be observed when selecting the elements that are to be defined in the subschema.

If an area's definition is not included in the subschema, groups specified as within the area may not be defined in the subschema. Nor may any set be defined in the subschema if its owner or any member is specified as within the area. In addition, if an invert group for secondary indexes was specified as within the area, either the associated item must be excluded from the subschema or it must be specifically selected with an indication that inversion is not to occur.

If a group definition is to be omitted from the subschema, all sets in which the group participates as owner or member must also be excluded. (All items in the groups are automatically excluded.)

Not all data manipulation capabilities are allowed when a subschema does not define a complete database. For example, a program may not store or delete occurrences of a group that is the owner or a member of a set that is not defined in the subschema the program is using; nor may it store group occurrences if the definition of any item in the group is omitted. Refer to the description of the DBM routines for more details on which are restricted when operating with a limited subschema.

The subschema DDL consists of entries in the following order:

1. The Subschema entry must be the first entry.
2. The set entry (there is only one) follows the subschema and precedes all area entries.
3. The area entry (or entries) for any areas to be included follow the set entry and precede all group entries.
4. The group entries follow the area entries. Group entries consist of a group subentry and, optionally, one or more item subentries.
5. The end entry must be last.

Subschema Entry

A subschema entry provides the name for the subschema file, specifies whether all or part of the database is to be defined by the subschema, and indicates the form of working storage declarations that are to be generated.

Format

```
SUBSCHEMA NAME IS sub-schema-name OF SCHEMA schema-name

  [; COBOL COPY    IS copy-name]

  [; META SYSTEM   IS system-name   [, NAMECHECK] ]

  [; PRIVACY KEY FOR EXTRACT IS privacy-lock]

  [{PASSWORD IS  } password-1[, password-2]...]
   {PASSWORDS ARE}

  ; COMPONENTS ARE {ALL      }
                  {SPECIFIED}
```

Usage Rules

1. The SUBSCHEMA NAME clause must be the first clause in the entry. Subschema-name is the file name by which the subschema file is to be referenced. Hence it must conform to the host operating system's file naming conventions as well as the DDL rules for names. The schema-name must be the file name of an existing schema file.

2. The COBOL clause provides the name for a COBOL source file that is to contain declarations that define the user's working storage needed for database operations based on this subschema. The copy-name must conform to the DDL rules for names as well as to the conventions of the host operating system.

3. The META clause provides the name for a Meta-Symbol source file that is to contain the directives needed to define the user's working storage that corresponds to this subschema. The system-name must conform to the DDL rules for names and to the conventions of the host operating system. The names of the groups, items, and sets selected for the subschema will be modified to conform to Meta-Symbol standards by replacing all hyphens with dollar signs.

   Additionally, if the NAMECHECK option is specified, a symbol consisting of the "at" character (@) followed by the group name will be appended to each item name to ensure uniqueness with regard to like-named items in other groups. If the NAMECHECK option is not specified, the user is responsible for ensuring that his item names are unique.

4. The PRIVACY clause supplies the key required to enable the generation of a subschema if the specified schema has a PRIVACY LOCK FOR EXTRACT attached to it. The specified privacy-lock must be a nonnumeric literal and must match the lock on the schema, or the subschema will not be generated.

5. The COMPONENTS clause specifies that either the whole database (ALL) or selected parts of the database (SPECIFIED) are to be defined in the subschema. SPECIFIED indicates that a set entry follows the subschema entry. ALL indicates that the only other entry is an End entry.

6. The PASSWORD clause allows the user to specify which passwords from the schema are to be included in the subschema. Passwords are specified as nonnumeric literals, and all passwords specified must have been previously defined in the schema DDL. If the PASSWORD clause is omitted, all passwords defined in the schema DDL are included in the subschema.

Set Entry

The set entry lists the sets that are to be defined in the subschema.

Format

```
{SET IS  }  {set-name-1    [,set-name-2] ...}  .
{SETS ARE}  {   ALL                          }
```

Usage Rules

1. The specified set names must be names of sets that are defined in the schema.

2. For each set listed, the owner and all the member groups must be defined in the subschema. The groups may be specified by group entries, or they may be implied by the COMPONENTS ARE ALL option on an area entry.

Area Entries

Area entries specify the areas of the database that are to be available through this subschema, and indicate whether all or part of the specified areas are to be defined. A single area entry may name several areas that

have the same components specification or a separate area entry may be included for each area. No area entries are allowed if the subschema entry specified COMPONENTS ARE ALL.

Format

```
{AREA IS  }  {area-name-1   [,area-name-2] . . .}
{AREAS ARE}  {ALL                               }

; COMPONENTS ARE  {ALL      }  .
                  {SPECIFIED}
```

Usage Rules

1. The AREA/AREAS clause must be the first clause in the entry. The area-names must be names that exist as area names in the schema. The naming of selected areas or ALL areas indicates that some portion of the groups that may occur in the areas will be defined in the subschema.

2. The COMPONENTS clause determines that either ALL or SPECIFIED groups and items identified for the specified areas in the schema are to be defined in the subschema. If COMPONENTS ARE SPECIFIED, group entries must be included for any of the areas' groups that are to be included in the subschema.

Group Entries

Group entries are used to select the groups that are to be defined in the subschema. No group selection is needed or allowed if no area entries indicated COMPONENTS ARE SPECIFIED. To be defined in the subschema, any non-invert group within an area whose components are specified must be selected by a group entry. Invert groups' definitions are automatically included if the secondary index item is defined in the subschema and inversion is not specifically suppressed.

Group Entry Skeleton

Group Subentry

[Item Subentry]. . .

Group Subentry

A group subentry identifies the group, optionally renames it (for working storage declarations), and indicates whether some or all of the group's items are to be defined in the subschema group defintion.

Format

```
GROUP NAME IS group-name-1 [, RENAMES group-name-2]

; COMPONENTS ARE {ALL      }  .
                 {SPECIFIED}
```

Usage Rules

1. The GROUP NAME clause must be the first clause in the entry. If the RENAMES option is not specified, group-name-1 must be the name of a group defined in the schema as within an area that is named in a

subschema area entry. If RENAMES is specified. group-name-2 must be the name of a group that is so defined. If RENAMES is specified, group-name-1 must conform to the DDL rules for names and must not duplicate the name of any group or set in the subschema.

2. The COMPONENTS clause specifies that either ALL of the items defined for the group in the schema are to be defined in the subschema (exactly as they are defined in the schema) or that item definitions are SPECIFIED in item subentries that immediately follow the group subentry. If a change in any one item definition is desired, then all of the data items must be described in item subentries.

Item Subentries

Item subentries designate and optionally rename the items that are to compose the group as defined in the subschema.

If the group subentry specified COMPONENTS ARE ALL, no item subentries are legal. If the group subentry indicated COMPONENTS ARE SPECIFIED, all items that are to be included must be described in item subentries.

Format

```
[ level-number]  data-item-name-1  [, RENAMES  data-item-name-2]
[; INVERSION  IS [NOT] REQUIRED]
[; CONDITION NAME IS condition-name-1
    { VALUE IS  } literal-1  [THRU literal-2][, literal-3 [THRU literal-4]]  ...]...
    { VALUES ARE}
```

Usage Rules

1. The level number is optional, and if omitted is assumed to be the lowest level number specified for the group, or 02 if no previous level number has been encountered. Item level numbers may have values in the range from 02 through 49. Usage of level numbers is syntactically consistent with that described in the ANS COBOL/LN Reference Manual, 90 15 00.

2. Data-item-name-1 must immediately follow the level number (or must be the first element in the entry if no level number is included). If RENAMES is not specified and data-item-name-1 is not defined in the schema as being part of the group, it is assumed that the user desires to superimpose data-item-name-1 over one or more data items which are so defined. In this instance, DMSFDP requires that the item subentry containing data-item-name-1 be followed by at least one item subentry containing a data-item-name which is defined as part of the group being analyzed and contains a level number higher than that of data-item-name-1. Note that this feature is included solely for the convenience of COBOL programmers and that data-item-name-1 may not appear as an argument in a DBM call.

3. The INVERSION clause, unless NOT is included, specifies that the invert group associated with the secondary index item identified by data-item-name-1 is to be defined in the subschema. This clause is legal only for data items which appeared as data-item-name-1 in an INVERT entry of the schema DDL. If the INVERSION clause is omitted, it is assumed that the invert group definition is required.

4. The CONDITION NAME clause causes a level-88 data description entry to be included in the COBOL Copy file. Condition-name-1 must conform to DDL rules for names. Literal-1, literal-2 etc. are numeric or nonnumeric literals depending on the schema definition of the item identified by data-item-name-1 and must conform to the size and form of the item.

END Entry

The end entry signifies the end of the subschema description.

Format

END.

DMSFDP Operational Interface

The File Definition Processor may be operated in a batch mode or from a terminal. The operation of DMSFDP relative to the amount and format of output is controlled by control command options. The control command has the following form:

!DMSFDP [, NODDL][, NOSCHEM][, NOSUB][, NOCBL][, NOMETA][, NOLIST][, NONAMES]

The order in which the options are specified is immaterial but repetition of an option is not allowed.

Exercising the options suppresses the normal output. The !DMSFDP with no options causes the following:

1. A schema will be created if the first DDL entry is a schema entry, no DDL errors are encountered, and there is not an existing file in the user's account that has the same name as that specified in the schema entry.

2. A subschema will be created if the first DDL entry or the first entry after a schema-DDL end entry is a subschema entry, no DDL errors are encountered, and the file name specified in the subschema entry is not the name of an existing file in the user's account.

3. All DDL entries will be listed (both schema and subschema entries, if both are included in one run).

4. All error messages and summary messages will be listed. Error messages include a $ character printed under the DDL line at the point where the error was detected and an explanatory message. Table F-1 in Appendix F shows the DMSFDP error messages. Summary messages include information on file size and structure plus number of diagnostic messages. A number of diagnostic messages other than zero indicates that the generated schema/subschema file was not saved. Figure C-2 in Appendix C illustrates the summary messages output by DMSFDP.

5. A COBOL COPY file will be created and its contents listed if the subschema entry includes a COBOL clause. Figure C-4 shows a simple COPY file listing.

6. A Meta-Symbol SYSTEM file is created and its contents listed if the subschema entry includes a META clause. Figure C-6 in Appendix C shows a sample SYSTEM listing output. The FORTRAN user may use either the COPY or the SYSTEM listing to determine the format of the working storage area to be declared in his program.

7. A name table relating set, group, and item names to their subschema definitions, is included in the subschema file.

The suppress options operate as follows:

1. NODDL – only erroneous DDL input statements are to be listed, correct DDL statements are not to be listed. This does not affect the listing of COBOL and META files.

2. NOSCHEM – the schema file is not to be saved. (This may or may not affect the creation of a subschema in the same run; the subschema creation requires a valid, existing schema file, but it may have been created on an earlier run.)

3. NOSUB – a subschema file is not to be saved. This affects only the subschema file; any listing or other file creation is controlled separately.

4. NOCBL – the COBOL COPY file is not to be created even though the subschema entry may include a COBOL clause.

5. NOMETA – the Meta-Symbol SYSTEM file is not to be generated even if the META clause is included on the subschema entry.

6. NOLIST – COBOL or Meta-Symbol data is not to be listed even if the corresponding file is created.

7. NONAMES – the subschema file is not to include the name table.

The NODDL option applies to both schema DDL and subschema DDL. NOSCHEM is obviously meaningless if only subschema DDL is input; it is therefore ignored. Similarly, NOSUB, NOCBL, NOMETA, NOLIST and NONAMES are meaningless if only schema DDL is specified.

DCB Assignments

Normally, no interface is required between the DMSFDP user and the CP-V monitor to create the schema, subschema, COBOL COPY, and Meta-Symbol SYSTEM files. The user may assign the M:SI and M:LO DCBs to accept the DDL input or to direct the listing output to other than the system standard devices.

The F:SCHE (schema), F:SSCH (subschema), F:COPY (COPY file), and F:META (SYSTEM file) DCBs may be assigned if desired. One or more such assignments might be needed, for example, to place the files on a removable device or, in the case of the schema, to specify WRITE accounts so that subschema generation can be run in an account different from that used to generate the schema. (Subschema generation involves writing into the schema file.)

Terminal Usage

DMSFDP may be run from a terminal with DDL either input directly or (preferably) stored in an EDIT file. In either case, the user initiates operation by entering DMSFDP in response to the system prompt and then entering the control command options (or carriage return if there are no options) in response to the prompt from DMSFDP.

4. DATABASE MANAGER

Database manager (DBM) is the term applied to the collection of library routines that are used with a user's applications program to accomplish the storage, retrieval, and updating of the data values and pointers in a database. Other features of the DBM provide for collecting run-time and summary statistics, tracing a user program's interaction with the DBM, maintaining a journal of changed pages, and recovering a shared database in case of deadlock or upon user request.

The user's program communicates with the DBM by means of calls to the library subroutines. Most arguments for the calls refer to addresses within the program's working storage, which must be formatted to correspond to the values in the subschema being used.

The user's program area that is referred to as working storage consists of two parts. The first part has the same format in all EDMS programs, regardless of the nature of the database used. The second part must be formatted to reflect the specific subschema referenced by a program. The first part of working storage is designated the Communication Control Block (CCB) because it is used to communicate control and current-condition information between the user program and the DBM. The format of the CCB is described in Table 2, which uses the COBOL COPY file form for identifying the contents. In the Meta-Symbol SYSTEM file the hyphens are replaced by dollar signs and the characters @CCB are appended, e.g., REF$CODE@CCB instead of REF-CODE.

The format of the database-specific part of the user's working storage must provide for a set table for each set defined in the subschema, a group table for each group defined in the subschema, a statistics table if any statistics are specified for the database, and a table for aliases if any are defined. The number and order of occurrence of these entities depend on the subschema being used. The proper order is best obtained by using or following one of the working storage descriptions generated by the File Definition Processor. Figure C-4 shows an example of the COBOL COPY working storage, and Figure C-6 shows an example of Meta-Symbol SYSTEM working storage, both generated for the sample database shown in Figure 1, but for separate subschemas. (The FORTRAN user may use either of the generated descriptions as a guide for manually generating declarations.)

The group tables are used to communicate item values and the reference code of the current occurrence of the group.

The set tables are used by the DBM to maintain the position of the user in each set. Each time a group occurrence is retrieved explicitly by the user or implicitly by the DBM, the set table for each set defined in the subschema for that group is updated. The address of the set table is used as an argument for set-processing DBM-routine calls in the same manner as group or item arguments.

DBM Routine Call Format

The DBM routines that store, retrieve, etc., are initiated by calls in the user's applications program. The format of the call depends on the language in which the user's program is written; but whatever the language, the call refers to a DBM function name, which is an entry point in the DBM library routines.

The general form used in the manual to describe the DBM calls is

ENTER DBM-function-name, argument-1[,argument-2]. . .

where the arguments represent addresses (optionally indirect) within the user's program area, either word addresses or byte addresses, depending on the programming language used and on the characteristics of the entity located at the specified address. In the descriptions of the DBM calls, below, the address arguments are referred to by descriptive terms. REF-CODE, area-name, item-name, group-name, and set-name denote addresses in the user's program areas that correspond to DMSFDP-generated working storage declarations; error-code-name and recovery-name denote addresses in the user's program area other than that corresponding to the DMSFDP-generated working storage declarations; and procedure-name denotes an address in the user's program area to which the DBM is to return control under certain conditions. The metalanguage used below to show DBM call formats is the same as that used to depict the Data Definition Language (see "File Definition Processor," Chapter 3).

Table 2. Contents of the Communications Control Block

Contents	Description
REF-CODE	A 32-bit binary number whose value is the reference code of the group last accessed by the user. At the successful completion of any call that accesses a group occurrence in the database, the reference code of the group is placed in this cell by the DBM. The reference code is also used when an area is opened to specify the number of buffers, when an area is closed to indicate whether or not core is to be released, and when a group occurrence is to be stored or retrieved directly.
PAGE-NO	Contains the eight-character EBCDIC value of the page-number part of the reference code. This value is supplied by the DBM at the successful completion of a call in the same manner as REF-CODE.
LINE-NO	Contains the three-character EBCDIC value of the line-number part of the reference code. This value is supplied by the DBM in the same manner as PAGE-NO.
FRST-REF	A communication cell used in conjunction with the FINDS or FINDSI procedural calls. The user must initialize this cell with the reference code at which the DBM is to start the physical scan of an area of the database.
LAST-REF	A communication cell used in conjunction with the FINDS or FINDSI procedural call. The user must initialize this cell with the value which will control the termination of the physical scan of an area of the database.
GRP-NO	Contains a 10-bit binary number whose value is the numeric synonym for the group stored or retrieved by the user.
ERR-CODE	Initialized by the DBM with an eight-bit binary number whose value indicates that some type of error occurred in executing the previous procedural call.
ERR-NO	A 10-bit binary number initialized by the DBM for certain types of errors, with the numeric synonym for the group responsible for the error.
ERR-REF	A 32-bit binary value initialized by the DBM for certain types of errors, with the reference code of the group responsible for the error.
PASSWORD	A communication cell that must be initialized by the user with the eight-character EBCDIC value of the password that allows the user access to the database.
AREA-NO	Contains the two-character EBCDIC value of the area number part of reference code. This value is supplied by the DBM in the same manner as PAGE-NO.

Meta-Symbol Call Format

A Meta-Symbol call takes the following form:

```
REF       DBM-function-name
LI, 14    n                      number of arguments
BAL, 15   DBM-function-name
* *       address-1
            :
* *       address-n
```

The asterisks indicate that the addresses are right-justified and may be generated by any of several Meta-Symbol techniques. The addresses supplied may be indirect (but not indirect in a register), in which case the DBM will obtain the proper effective address, either word-oriented or byte-oriented as shown below in Table 3. The examples in Table 3 are from a Meta-Symbol program that includes the SYSTEM file shown in Figure C-6, and processes part of the data base shown in Figure 1.

FORTRAN Call Format

A FORTRAN program call of a DBM library subroutine takes the form of a standard calling sequence, as follows:

CALL DBM-function-name (argument-1, ...)

The arguments used must result in addresses supplied to the DBM that conform to the DBM function description shown in Table 4. All addresses are word addresses.

COBOL Call Format

The call from a COBOL program provides the model for the form of the DBM function description. It takes the form of the ENTER statement.

ENTER DBM-function-name[,argument-1]...

Arguments to the ENTER statement of COBOL are either the data names of the appropriate data segment in the data division or the procedure name in the procedure division.

Table 5 shows the values of the arguments to generate the types of addresses required. The examples reference the COPY file names shown in Figure C-4.

Table 3. Meta-Symbol Addresses

If the DBM Function Description Specifies	The Address Supplied Must Be
REF-CODE	Word address of the first word of the CCB. WA(REFSCODE@CCB).
Area-name	Byte address of the appropriate area name word of the area table. For example, BA(AREA$2).
Group-name	Byte address of the first word of user's working storage reserved for the group. For example, BA(FIRST).
Item-name	First byte of working storage reserved for the item, if the item is EBCDIC or packed decimal; first word of working storage reserved for the item if the item is binary or floating point. For example, BA(ITEM$31), but WA(ITEM$44).

Table 3. Meta-Symbol Addresses (cont.)

If the DBM Function Description Specifies	The Address Supplied Must Be
Set-name	Byte address of the first word of the user's working storage reserved for the set table of the set. For example BA(SET$D).
Procedure-name	Word address of the location to which the DBM is to return control.
Error-code-name	Word address of a location in the user's program area that contains an EDMS data-dependent error code in binary.
Recovery-name	Word address of a location in the user's program area that contains the EBCDIC characters RECV.

Table 4. FORTRAN Addresses

If the DBM Function Description Specifies	The Argument Must Be
REF-CODE	The identifier of the first variable in EDMS working storage.
Area-name	The identifier of the variable used to establish to appropriate area entry.
Group-name	The identifier of the first variable used to reserve working storage for the group.
Item-name	The identifier of the appropriate item variable.
Set-name	The identifier of the first variable used to reserve working storage for the set s table.
Procedure-name	A statement label.
Error-code-name	The identifier of a location established by the user. The value in the location must be between 1 and twenty, inclusive.
Recovery-name	The identifier of a location established by the user. The value in the location must be the Hollerith constant RECV.

Table 5. COBOL Arguments

If the DBM Function Description Specifies	The Argument Used Must Be
REF-CODE	Data-name REF-CODE of the CCB.
Area-name	The name assigned to the area in the DDL. For example, AREA-1.
Group-name	The data-name of the 01 level entry of the group. For example, GROUP-1, GROUP-2-R.
Item-name	The data-name of the appropriate item. For example, ITEM-21-22-23.
Set-name	The data-name of the 02 level entry generated for the set table for the set. For example, SET-C.
Procedure-name	A name in the procedure division.
Error-code-name	The data-name of an entry generated by the user. The entry must be COMP usage and have a VALUE of 1-20.
Recovery-name	The data-name of an entry generated by the user. The entry must be alphabetic or alphanumeric and contain the value 'RECV'.

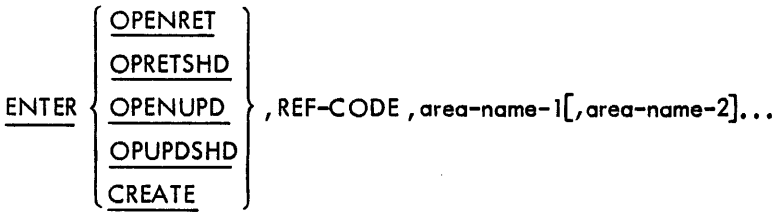
DBM Routine Usage

Database manager routines are used to accomplish all user-program interaction with the database. The first step of a user-program's interaction is to open all areas that are to be accessed by the program. After all required areas are opened and depending on the type of open, new group occurrences may be added to the database, obsolete data may be deleted, data values or set linkages may be modified, existing group occurrences may be retrieved, and various miscellaneous functions may be performed by calling the appropriate DBM routines. The last DBM call from a user program is to close the areas (or the last area in use) of the database to terminate processing. All of these interactions are described below.

Beginning of Processing

Before any data manipulation activity can occur, the files in which the data is stored must be opened. The DBM interacts with the operating system to open the file in response to an open-call from the using program. The open-call identifies the area to be opened, and indicates what type of activity is intended.

Format



Usage Rules

1. An area must be opened before any other EDMS call that references the area (either directly or indirectly) is executed. A call to open an already opened area is ignored, if no calls other than open calls are made between the two opens.

2. A call to open an area may not be made if the user is currently executing in some other area, i.e., there may be two or more successive calls to open different areas only if there are no other intervening procedural calls that reference the first area.

3. OPENRET opens an area for retrieval purposes only. Other programs may concurrently open the area in OPENRET and OPENUPD mode. The user should be aware that this mode does not provide for protection against changes made to the database by another program concurrently executing in the OPENUPD mode.

4. OPRETSHD opens an area for retrieval purposes and specifies that the area may be accessed concurrently by other programs in this mode or for shared update.

5. OPENUPD opens an area for both retrieve and update purposes. Other programs may concurrently open the area in OPENRET mode only.

6. OPUPDSHD opens an area for retrieve and update and specifies that the area may be accessed concurrently by other programs in this mode or for shared retrieval.

7. If any areas are opened in a shared mode (OPRETSHD or OPUPDSHD) by a program, no other areas may be concurrently opened in a non-protected mode (OPENRET, OPENUPD, or CREATE) by the program.

8. CREATE is a special open mode for an area that has a group defined with location mode of indexed. While an area is open in CREATE mode, the key values of an indexed group occurrence to be stored must be higher than those of the occurrence most recently stored; i.e., the group occurrences must be presented to the DBM for storage in ascending key order (see "Adding Occurrences", below). The area may be concurrently opened in OPENRET mode by other programs.

9. For all open modes, REF-CODE refers to the address of the beginning of the user's formatted working storage. This location should contain the number of data buffers to be used (3 to 10, inclusively) at the time the first open call is made. If a number less than 3 is specified, 3 will be used; if a number greater than 10 is specified, 10 will be used.

10. If any passwords were specified for the database, the call PASSWORD in the CCB must be initialized before an open call is made. An eight-character password that is associated with keys that allow access to the desired groups and items should be supplied.

11. If an area to be opened is an enciphered area, the user must supply the enciphering key in the appropriate area-name cell prior to the open call.

12. If any area of the database has been closed and is to be reopened, all areas must first be closed; i.e., reopening an area may not violate usage rule 2 above.

DBM Response

If any one of the required parameters is not supplied in the CCB; if any of the named areas is not assigned; if processing has begun in an area; or if mixed mode (shared and non-protected) opens are attempted, the DBM returns an error indication in ERR-CODE in the CCB. If all conditions are satisfactorily met, the DBM sets up the controls necessary for processing the areas. The area files are not opened until a subsequent DBM call references one of the areas.

When an area is opened in exclusive mode (OPENRET, OPENUPD, or CREATE), no provision is made for dynamic recovery in case of deadlock, because deadlock cannot occur, and there is no requirement for locking of individual pages.

When an area is opened in shared mode (OPRETSHD, OPUPDSHD) individual pages are locked, by means of the CP-V enqueue/dequeue facility, as required (see Appendix H for additional information on Enqueue/Dequeue).

If an area is reopened, the DBM will zero out the contents of the set tables and current-of-type for all sets and groups defined for the area. Thus, a program may not maintain a logical position in an area between close and open.

Adding Occurrences

The first activity involving the data in the database is to load, or store, group occurrences in the area files. This activity continues with varying frequency over the life of the database. The required conditions and the action of the DBM when a request is made to add a group occurrence to a database area depends on a variety of factors such as whether or not inventory pages exist for the area, whether or not there is an indexed group within the area, what the location mode of the group is, what sets it participates in and how, etc.

Format

ENTER STORE, group-name

Usage Rules

1. The data values that are to constitute the group occurrence should be in the working-storage designated for the group.

2. If the group is an automatic member of any set, the desired set occurrence must be selected. This is done either by retrieving the owner occurrence or a member occurrence if the set selection is current (unnecessary if the occurrence most recently stored or retrieved is part of the desired set occurrences), or by putting the uniqueness-determining value in working storage if selection is through location mode of owner. Note that the uniqueness-determining values may be calc keys, index keys, set sort keys, or a combination of sort keys and one of the others if several levels of owner are required to establish uniqueness.

3. STORE is not permitted if the specified group is the owner or a member of a set that is not defined in the subschema being used; if any item in the group is not defined in the subschema; if the subschema item subentry for a secondary index item specified no inversion; or if the group is a member of a multimember sorted set without group number as major, and the definition of a sort key item in one of the other member groups is not included in the subschema.

DBM Response

The DBM must physically and logically position the occurrence in the area. To physically position the occurrence, the DBM determines a base page for the occurrence and stores it on that page if there is space. The base page for a group occurrence is determined differently for each location mode as follows:

CALC – The values of the calc control items are randomized across the page range for the group to determine the base page.

INDEXED – The values of the index control items are compared to the primary index entries. The base page for the group occurrence is that page which contains the occurrence of the group that has the next-higher values in its index control items. If no occurrence of the group currently in the area has higher values, the base page is the last page currently containing indexed group occurrences.

DIRECT – The base page is provided by the user in cell REF-CODE. If a storage parameter is selected for a direct group, the base page is determined as if the group were a via group.

VIA SET – The base page is determined by the order of the via set, or storage set if appropriate, and the existing members of the set:

a. Sorted – base page is the data page of the current group occurrence logically before the new occurrence in its sorted sequence.

b. First and Last – base page is the data page of the set owner occurrence.

c. Next and Prior – base page is the data page of the current member occurrence of the set.

If there is not sufficient space, or no available line number on the base page, the DBM systematically searches until space is found, or if no space can be found, the DBM returns an error code in the CCB. The search is based on the location mode of the group and whether or not there is an indexed group and an overflow range in the area.

If the occurrence cannot be stored because of subschema limitations, if the password supplied at open does not provide an update key required for storing occurrences of the group; if values in the occurrence are duplicates of values for which duplicates are not allowed (calc keys, sort keys, secondary indexes for which duplicates are not allowed, or indexed location mode keys); if key values for an indexed group are not in ascending order in create mode; if any values do not meet data validation criteria; or if a deadlock is precipitated during the store processing, the DBM returns an error code in the CCB. Additional action is taken in the case of deadlock (see "Preparing for Deadlock", below).

The group occurrence is logically positioned in all sets in which it participates according to the set selection and the set order. The occurrence is linked into all sets in which it is an automatic member. If the occurrence cannot be linked for some reason e.g., the correct owner occurrence cannot be retrieved, the DBM returns an error code in the CCB.

At the successful conclusion of a STORE call, the group occurrence is recorded as

Current-of-file – Assigned reference code is in REF-CODE of CCB.

Current-of-type – Assigned reference code is in the CURR-XXX cell in user's working storage.

Current-of-set – Assigned reference code is in SET-CURR of all sets of which the group is an owner or automatic member.

The numeric synonym for the group is also placed in GRP-NO of the CCB.

Deleting Occurrences

A group occurrence can be physically removed from the database or marked as unavailable and flagged for future removal, or the delete call can specify conditions under which the group is to be deleted. If the subschema being used does not describe the complete database, there may be some EDMS-imposed restrictions on deleting group occurrences.

Format

ENTER { DELETE / REMOVE / DELETSEL / REMOVSEL / DELETAUT }, group-name

Usage Rules

1. The group occurrence to be deleted is the occurrence identified as current-of-type for the group named.
2. The occurrence cannot be deleted if any set of which the group is an owner or member is not defined in the subschema or if any invert group associated with the group is not defined in the subschema.
3. The occurrence cannot be deleted if some member group at a lower level cannot be deleted because of subschema omissions.

DBM Response

If the occurrence cannot be deleted because of subschema limitations; if the password supplied at open does not provide update keys for one or more of the groups affected; if the current-of-type is not established, or if the delete processing precipitates a deadlock, the DBM returns an error code in the CCB (there is additional processing in the case of deadlock, see "Preparing for Deadlock" below).

If necessary conditions are met, the response is as follows:

DELETE – The group occurrence and any associated member group occurrences in a set of which it is the owner are logically deleted from the database. The deleted group occurrences will only be physically removed from the database if this does not require examining a complete set to establish the prior occurrence of the deleted group.

REMOVE – The group occurrence and all of its associated member occurrences are logically and physically removed from the database.

DELETSEL – The group occurrence is logically removed from the database only if it does not have associated member occurrences. If the group occurrence is the owner of a nonempty set occurrence, the DELETSEL call is not executed and an error code is returned in the CCB.

REMOVSEL – The group occurrence is logically and physically removed from the database only if it does not have associated member occurrences. If the object group is the owner of a nonempty set occurrence, the REMOVSEL call is not executed and an error code is returned in the CCB.

DELETAUT – The group occurrence is logically deleted from the database. If the group is defined as the owner of a set with automatic members, all automatic-member occurrences will be logically deleted from the database. Any deleted automatic-member occurrences will be treated as if they were the object of a DELETAUT call. If the group is defined as the owner of a set with manual members, the manual-member occurrence will be delinked from the set. Execution of this call makes all deleted group occurrences unavailable for subsequent access by the user. The current-of-type for groups whose occurrences are deleted and the current-of-file (REF-CODE) are set to zero.

Modifying Data Values

The values of one or more items in a single group occurrence can be modified.

Format

ENTER MODIFY, group-name [, item-name]...

Usage Rules

1. Before executing this call, the user must initialize working storage with the new values for the items to be modified.

2. The object of the call is the group occurrence that is current-of-type for the group named.

3. The list of item-name arguments identifies the specific items to be modified. If no list is given, it is assumed that all defined items in the group are to be modified.

4. This call may not be used under any of the following conditions:

   a. If the item is a calc control item and definitions of other calc control items are omitted from the subschema.

   b. If the item is a sort key for a set and definitions of other sort keys from the same set are omitted from the subschema.

   c. If the item is a sort key and the definition of the sorted set is omitted from the subschemas.

   d. If the item is a sort key for a multimember set sorted without group number as major sort key and the definitions of the sort keys in the other member groups are not all included in the subschema.

   e. If the item is a secondary index and the invert-group definition is omitted from the subschema.

   f. If the item is an indexed location mode control item.

DBM Response

If one of the above conditions is not met; if the password supplied at open does not provide an update key required for modifying the item(s); if a new value duplicates an existing value for which duplicates are not allowed (a calc key, sort key or secondary index item with no duplicates); if the current-of-type for the group has not been established (e.g., by a previous retrieval or store action); or if the modify attempt results in a deadlock with another program, the DBM returns an error code in the CCB. (Additional actions in the deadlock case are described under "Preparing for Deadlock", below.)

If there are no errors, the item value(s) is replaced with the new value(s).

If an item to be modified is a calc control item for the group, the item values are changed and the pointers in the group occurrences affected are modified to indicate the new base page. The group occurrence with the modified value, however, is not physically moved to the new base page. If an item to be modified is a sort control item for a set in which this group is a member, the item values are changed and the group occurrence logically repositioned in the set based upon the modified item values.

Modifying Linkages

An occurrence of a group whose membership in a set is defined as optional or manual can be linked to or delinked from a set occurrence (LINK and DELINK). Also, a member group occurrence can be changed from one owner occurrence to another in any set in which it participates (RELINK).

Linking, Delinking, or Relinking Member Occurrences

Format

ENTER { LINK / DELINK / RELINK }, group-name, set-name

Usage Rules

1. The object of the call is the group occurrence that is current-of-type for the group named.

2. To DELINK a group occurrence from the named set, the group must be defined as an OPTIONAL member or a MANUAL member, and the occurrence must be linked into a set occurrence.

3. To LINK a group occurrence into the named set, the group must be defined as a MANUAL member or an OPTIONAL member, and the occurrence must not be currently linked into a set occurrence.

4. To RELINK a group occurrence from one occurrence of the named set into another, the group must be defined as a member of the named set, and the object group occurrence must be linked into an occurrence of the set.

5. For LINK or RELINK, the set occurrence into which the object group occurrence is to be linked must be selected. If the defined set selection technique is through location mode of owner, working storage must be initialized with the control-item values that uniquely identify the owner occurrence. If set selection is through current-of-set, the set occurrence should be established as current by means of a DBM call. This would normally be done by retrieving the owner occurrence or an occurrence of a different group type that is also defined as a member of the set. For RELINK, the current set occurrence should not be established by retrieving an already-linked occurrence of the named group, because that would make the already-linked occurrence current-of-type and the object of the call, which is contrary to the purpose of the call, and is effectively a null action.

DBM Response

If any of the above conditions is not met; if the password supplied at open did not provide update access to the named group; if processing the call would result in non-allowed duplicate values of sort-keys; or if deadlock with another program occurs, the DBM returns an error code in the CCB (additional action in the case of deadlock is described under "Preparing for Deadlock", below).

If the LINK call is successful, the object group occurrence is current of the named set.

If the DELINK procedure is successful, the group occurrence that was prior to the object group occurrence is current of the named set.

If the RELINK is successful, the object group occurrence is delinked from its previous set occurrence and linked into the new one. The DBM will not check to determine that the new set occurrence is indeed different from the previous set occurrence. If the order of the named set is sorted, the DBM will initialize working storage with the values of the sort control items from the object group occurrence to ensure that the object group occurrence is relinked into the proper logical position in the new set occurrence.

Retrieving

Various techniques are used for retrieving specified group occurrences from the database and making them available in the buffers. (Subsequent GET calls must be made to move the data into user's working storage.) The selection of the technique depends upon the specific application. Technique selection must be governed by the group and set characteristics of the occurrences to be retrieved. A single general format applies for the various techniques.

Format

```
ENTER { FINDG, group-name
      | FINDC, group-name
      | FINDD
      | FINDM, set-name
      | FINDN, { set-name
               | group-name, procedure-name }
      | FINDP, { set-name
               | group-name, procedure-name }
      | FINDS procedure-name
      | FINDSI, procedure-name
      | FINDX, group-name, item-name, procedure-name
      | FINDSEQ, group-name, item-name, procedure-name
      | FINDFRST, group-name
      | FINDLAST, group-name
      | FINDDUP, group-name }
```

Usage Rules

1. In each form of the retrieve (FIND) call, it is assumed that any data items necessary to identify the specific occurrence of the group to be retrieved have been initialized in working storage. The data items that are necessary depend on the specific call and are described under "DBM Response", below.

2. FINDG will not be allowed if

   a. Calc or index control items for the group are not defined in the subschema.

   b. The via set is not defined in the subschema.

   c. The via set is defined and one or more sort keys are not defined in the subschema.

   d. The via set is sorted without group numbers as major, and sort keys of another member group are not defined in the subschema.

3. FINDG is also not allowed for the area group established to function as set owner.

4. FINDDUP is not allowed if any of the calc control items for the group are not defined in the subschema.

5. FINDX and FINDSEQ are not allowed if the invert-group is not defined in the subschema.

DBM Response

1. The action in each case causes the group occurrence to be made available in one of the DBM buffers. No other action, such as moving the group to working storage, is implied.

2. At the successful conclusion of any retrieve call except FINDX, the object group occurrence is recorded as follows:

   Current-of-file -- The reference code of the group occurrence is stored in the REF-CODE entry of the CCB.

   Current-of-type -- The reference code of the group occurrence is stored in the CURR-XXX entry of user working storage (XXX is the numeric synonym for the group).

   Current-of-set -- The reference code of the group occurrence is stored in the SET-CURR entry of the set tables for each set in which the group participates.

   Group-type -- The numeric synonym for the group whose occurrence is retrieved is stored in the GRP-NO entry of the CCB. When using any retrieve call that does not explicitly identify the group name, an occurrence of any of several groups may be retrieved depending on the data structure involved. After execution of the procedure, the user program may determine the group whose occurrence was retrieved by referring to the GRP-NO entry of the CCB.

3. FINDG -- The FINDG (find-group) call retrieves a specific occurrence of the named group. The group occurrence retrieved is a function of the location mode of the group. When the group is defined as direct, the occurrence retrieved is identified by the reference code stored in the REF-CODE entry of the CCB. When the group is defined as calc, the occurrence retrieved is identified by the randomizing procedure, using the values of those items defined as randomize control items. When group is defined as via set, the occurrence must be retrieved via the owner occurrence of the set. In this last case, the values that uniquely identify the owner occurrence must have been initialized in working storage in addition to the values of those items (which must be SORT KEY items) that uniquely identify the via group occurrence. When the group is defined as indexed, the occurrence retrieved is identified by referencing the primary index to find the "base" page for the group and then using the values supplied for those items defined as the index items for the group to search the page set.

4. FINDC -- The FINDC (find-current) call retrieves the group occurrence identified by the reference code currently stored in CURR-XXX, where XXX is the integer identifier of the group named. This call is used to again retrieve the current-of-type group occurrence.

5. FINDD -- The FINDD (find-direct) call retrieves the group occurrence identified by the reference code stored in the REF-CODE entry of the CCB. If there is no occurrence with the specified reference code, or if the occurrence has been logically deleted, the DBM returns an error code in the CCB.

6. FINDM -- The FINDM (find-master-of-set) call retrieves the owner group occurrence of the set named. The action of this call depends on the contents of the set table for the named set.

7. FINDN -- The FINDN (find-next) call retrieves the next group occurrence in logical sequence of the set named if the argument to the call is a set name. The actual group occurrence retrieved depends on the user's position in the set as indicated by the set table.

   If the argument to the call is a group name, the group must be an indexed group and the call retrieves the group occurrence with the next higher key value. If, prior to the call, the user is positioned at the group occurrence with the highest key value, no group occurrence is retrieved, and control is returned to the user at the address specified by the procedure-name.

8. FINDP -- The FINDP (find prior) call retrieves the prior group occurrence in logical sequence of the set named if the argument to the call is a set-name. The actual group occurrence retrieved depends on the user's position in the set as indicated by the set table.

   If the argument to the call is a group name, the group must be an indexed group and the call retrieves the group occurrence with the next-lower key value. If, prior to the call, the user is positioned at the group occurrence with the lowest key value, no group occurrence is retrieved and control is returned to the user at the address specified by procedure-name.

9. FINDS – The FINDS (find-serial-search) call provides for a serial search of an area for the first group occurrence that falls within a range of reference codes. The range is defined by the user by the initialization of both the FRST-REF entry of the CCB with the first reference code of teh range and the LAST-REF entry of the CCB with the last reference code of the range. Control is returned to the user with each group occurrence found within the range after the DBM has incremented the value of the FRST-REF. Repeated execution of the call causes retrieval of each group within the range until the value of FRST-REF exceeds the value of LAST-REF. At this point, the call exits to the address specified by procedure-name.

10. FINDSI – The FINDSI (find-serial-search-from-initial-reference) call operates in the same manner as FINDS except that search limits are defined in terms of an initial reference code in FRST-REF and a number of group occurrences in LAST-REF. With FINDSI, the LAST-REF value is decremented with each group retrieved and the call exits to the address specified by procedure-name either when the LAST-REF value reaches zero or when the end of the area is reached.

11. FINDX – The FINDX (find-indexed) call locates and places into REF-CODE the reference code of the first group occurrence that contains a value (of the item named) equal to the value in working storage for that item. This call is only valid when the item-name has been defined as a secondary index (invert) item for the group named. Return from this call is to the first statement following the call when a group occurrence is identified that contains the value supplied in working storage. To find all group instances that match, the call must be used repeatedly within a loop without changing the value of the item in working storage. When no matching instances are found or when no additional instances exist, control is returned to the location specified by procedure-name. Any time the value of either the item in working storage or the FINDX arguments is changed, the DBM assumes that a new retrieval loop is involved and identifies the first matching group occurrence. Unlike other types of retrieval calls, FINDX does not actually retrieve the identified group occurrence. The only action apparent to the user program is the availability of the reference code of the qualifying data group occurrence in the CCB entry REF-CODE. Should the user wish to retrieve the selected group, he may do so by using the FINDD call.

12. FINDSEQ – The FINDSEQ (find-sequential) call sorts all occurrences of the specified secondary index (invert group) and serially retrieves the main group occurrences that correspond to the sorted invert group occurrences. This call is only valid when the named item is defined as an inverted item for the group named. The initial use of this call with a given set of arguments causes the DBM to build a sort input file consisting of all occurrences of the invert group for the secondary index, specified by item-name. The DBM then relinquishes control to the Sort processor, which sorts the invert group occurrences on the values of the invert item. At the completion of the sort, the DBM regains control, reads the first sorted invert group occurrence, retrieves the corresponding main group occurrence, and updates the CCB and set tables, as appropriate. Control is then returned to the first statement following the FINDSEQ call. Subsequent use of the call results in the retrieval of the next sequential main group occurrence until an end of file is reached on the sorted file, at which point control is returned to the location specified by procedure-name. Any time group-name or item-name is changed, it is assumed that a new sort is involved and the above-described initial procedure is executed.

13. FINDFRST – The FINDFRST (find-first) call retrieves the logically first indexed group occurrence, that is, the group occurrence with the lowest key value. This call is only valid when the group named has a location mode of indexed.

14. FINDLAST – The FINDLAST call retrieves the logically last indexed group occurrence, that is, the group occurrence with the highest key value. This call is only valid when the group named has a location mode of indexed.

15. FINDDUP – The FINDDUP (find-duplicates) call retrieves the next calc group occurrence that has randomizing control values equal to the current contents of user's working storage. This call is only valid when the group named has a location mode of calc and duplicates are allowed.

    Prior to this call the user must have retrieved a calc group whose randomizing control values are equal to the current contents of user's working storage. To execute this call, the DBM will find the next group of the calc set looking for a group with duplicate values. If none is found, an error will be returned in ERR-CODE of the CCB.

16. If the password supplied at open does not provide all necessary retrieve keys; if the values supplied in working storage are not sufficient to identify an occurrence; or if processing the call resulted in deadlock with another program (see "Preparing for Deadlock" below), the DBM returns an error code in the CCB.

Moving to Working Storage

The FIND calls only cause the page containing the selected group occurrence to be placed .. the buffer and the current indicators to be updated for the group and for the sets in which it participates. If the user wants to process the data in the group occurrence, the program must make an additional call. The GET call is used for this purpose. The HEAD call may be used to both retrieve and move a set owner-occurrence.

GET Call

Purpose. To move a retrieved group occurrence to working storage.

Format

ENTER GET, group-name [, item-name]...

Usage Rules

1. The object of the GET call is the group occurrence identified as the current-of-type for the group named.
2. The items to be moved to working storage may be any items defined within the group.
3. The list of item-names identifies the specified items to be moved. If no list is given, it is assumed that all items are to be moved.

DBM Response

The data values in the group occurrence are moved to working storage.

HEAD Call

Purpose. To both retrieve and move to working storage the owner group occurrence of a set occurrence.

Format

ENTER HEAD, set-name

Usage Rule

Before using this call, a previous database reference must have been made to establish a group occurrence as SET-CURR for the named set.

DBM Response

This call provides a function similar to the FINDM and GET calls except for the manner in which the set tables are updated. After execution of the HEAD call, the owner group occurrence is established as the current-of-type and as current-of-set for those sets in which the group is a member. It is not established as current-of-set for those sets in which it is owner.

Run-Time Statistics

Purpose. To initiate and terminate, by calls to the DBM, the collection of statistics on the performance of a program as it accesses a database. The statistics reflect the activity of that job only.

Format

ENTER { DMSSTATS / ENDSTATS / RPTSTATS }

Usage Rule

Run-time statistics collection can be initiated at any time during the operation of the program.

DBM Response

1. DMSSTATS causes the DBM to collect statistics on the activity of the specific job within the database. Statistics include the number of EDMS calls executed, the number of groups accessed by call and the number of physical page I/Os.

2. ENDSTATS causes the collection of the above statistics to be discontinued.

3. RPTSTATS causes a report of the statistics to be printed. After the report is written, the internal DBM counters for the statistics are reset to zero. A sample of the report is given in Figure 8.

Run-Time Tracing

Two types of trace information are accumulated by the DBM. The first type is initiated and terminated at the request of the user program and produces printed output. The second type is automatically maintained by the DBM and is not output.

User Initiated Trace

Purpose. To record and print the access record of DMS calls made by a program during program operation. (Listing output can be assigned to a file and printed later.)

Format

ENTER { DMSTRACE / ENDTRACE }

Usage Rule

The trace can be initiated and terminated at any time during the operation of the program being tested.

DBM Response

1. DMSTRACE causes the DBM to print the following information in its order of occurrence:

   DBM function name and user's calling address.

   Group number of group accessed and reference code of the occurrence.

   Number of page reads and writes.

   A sample of the trace is given in Figure 9.

2. ENDTRACE terminates the trace reporting.

```
PROCEDURE    CALLS    GROUPS
 FINDC           9         9
 STORE          85       381
     PAGE READS            6
     PAGE WRITES           1
```

Figure 8. Run-Time Statistics Sample

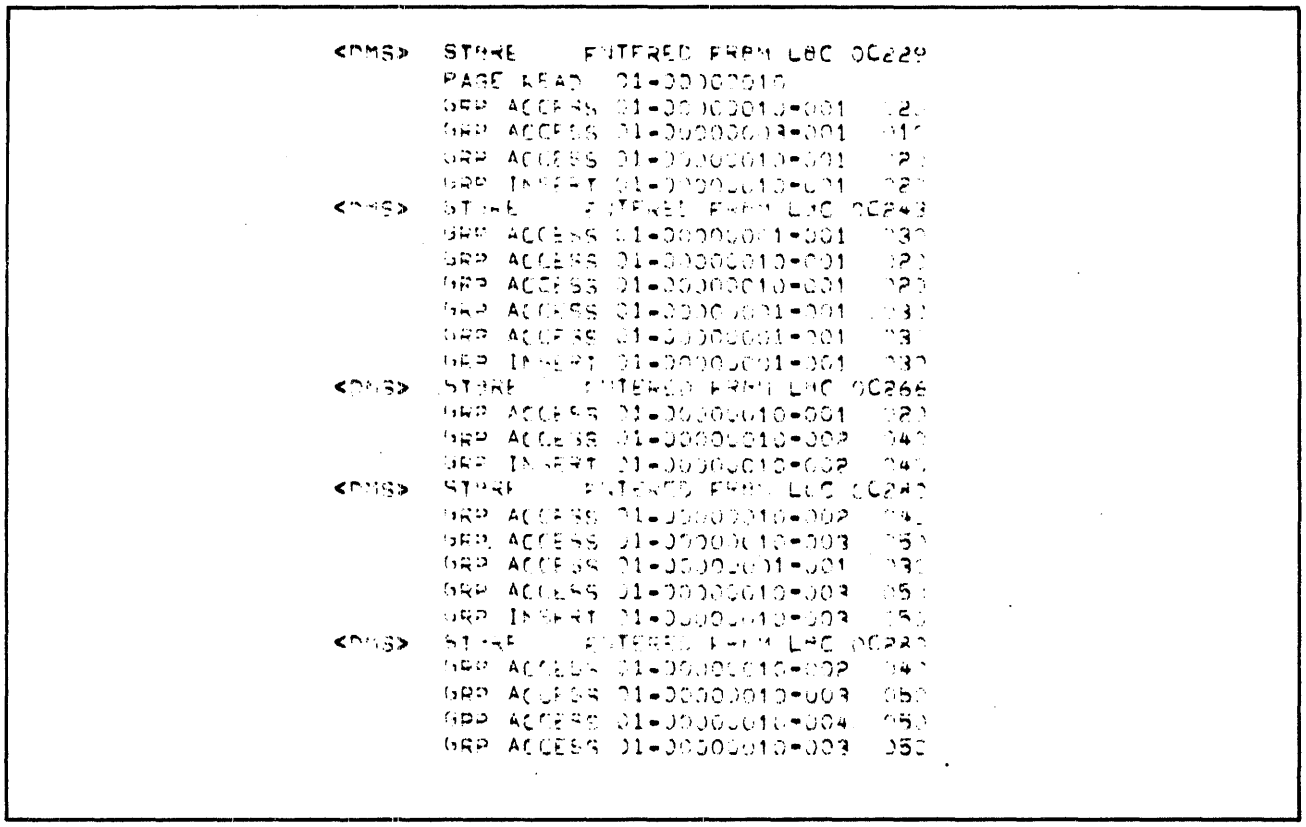


Figure 9. Run-Time Trace Sample

DBM Trace Table

The DBM maintains a record of user's calls in a trace table. No user action is required to initiate or terminate the maintenance of the table, nor can the table be displayed. The table may be examined in a memory dump or by using monitor SNAP commands. The trace table is a circular list of ten entries, controlled by a stack pointer at DEF Q:TRCTBL in the DBM. The table itself immediately follows the stack pointer doubleword, whose first word will contain the address of the current trace entry in the circular list. A trace entry has the following format:

bits 0-7 – binary value of an error code or zero.

bits 8-14 – binary code for type of DBM call (see Table 6).

bits 15-31 – address in the user's program from which the call was made.

Table 6. Trace Codes for DBM Calls

1. OPENUPD	17. GET	33. FINDM
2. OPRETSHD	18. MODIFY	34. HEAD
3. OPENRET	19. LINK	35. DMSRLSE
4. OPUPDSHD	20. DELINK	36. DMSCHKPT
5. CREATE	21. RELINK	37. CLOSEDB
6. CLOSAREA	22. STORE	38. FINDD
7. DELETE	23. FINDN (group)	39. DMSRETRN
8. DELETAUT	24. FINDP (group)	40. DMSTRACE
9. DELETSEL	25. FINDSEQ	41. ENDTRACE
10. REMOVE	26. FINDX	42. DMSSTATS
11. REMOVSEL	27. FINDS	43. ENDSTATS
12. FINDC	28. FINDSI	44. RPTSTATS
13. FINDG	29. Not Used	45. DMSABORT
14. FINDDUP	30. Not Used	46. SETERR
15. FINDFRST	31. FINDN (set)	47. RESETERR
16. FINDLAST	32. FINDP (set)	48. DMSLOCK

Error Control

Purpose. To enable the user's program to maintain a degree of control over the handling of DBM-detected errors by issuing a call that specifies a location to which the DBM is to return control in the event of a specified error condition.

Format

```
          ⎧SETERR, procedure-name [, error-code-name]...⎫
          ⎪RESETERR [, error-code-name]...              ⎪
ENTER     ⎨DMSRETRN                                     ⎬
          ⎪DMSABORT, procedure-name                     ⎪
          ⎩DMSLOCK, procedure-name                      ⎭
```

Usage Rule

All locations specified by procedure-name must be within the user's program area.

DBM Response

1. SETERR – Establishes the location that is to receive control in the event of a data-dependent error (codes 1-20). If no error-code-name arguments are given, procedure-name will receive control on any data-dependent error. If SETERR is entered with an error-code-name value that already has a procedure-name established for it, the new procedure-name will replace the previous one.

2. RESETERR – Disassociates a data-dependent error code value from a procedure-name so that the DBM will no longer trap to that procedure name if the error is encountered. If no error-code-names are given, all error code values are dissassociated.

3. DMSRETRN – Causes control to be returned to the statement immediately following the last DBM function call that resulted in an error for which the user had established an error-control procedure. The DMSRETRN call is used to exit from a procedure established by the SETERR call. The DBM will only retain the address of the last function call that resulted in an error.

4. DMSABORT – Establishes the location that is to receive control in the event of a non-data-dependent error other than deadlock (codes 31-137). The location established to receive control should be a wrapup routine as no additional DBM calls will be allowed.

5. DMSLOCK – Establishes the location that is to receive control if it causes a deadlock (error code 30) with another program that is sharing an area.

Preparing for Deadlock

There is a possibility of deadlock whenever two or more programs are concurrently accessing the same area, if at least one of them is updating the area (i.e., at least one program has used OPUPDSHD to open the area and at least one other program has used either OPRETSHD or OPUPDSHD). The deadlock occurs when two programs are each waiting for the other to release a locked page in order to proceed. An example is: Program A reads page 1 causing it to be locked with shared status. Program B then also reads page 1 locking it with shared status (many programs may lock a page with shared status without interfering with each other). Program A then attempts to update page 1, resulting in a request to promote the lock status to exclusive. This promotion is delayed waiting for Program B to remove the shared lock on page 1. If, instead of removing the shared lock on page 1, Program B also attempts to promote to exclusive lock status to update page 1, it will be delayed, waiting for Program A to remove its shared lock. The two programs are in deadlock and neither can proceed.

The monitor Enqueue/Dequeue function will detect a deadlock situation and return an error code to the program that finally caused the deadlock (Program B in the above example). The DBM will recover the database using any before images on the program's transient journal, thus undoing the program's database changes back to its most recent DMSRLSE call, or back to the beginning of its operation, if there was no DMSRLSE.

DMSRLSE Call

Purpose. To release pages that are locked for the program and make them available for reading and/or updating by other concurrently operating programs. The DMSRLSE call also establishes a point in the sequence of a program's operation as a base point for recovery in case of deadlock. The call notifies the DBM that some defined portion of the program's logic and/or input data has been completed, and that only subsequent database changes should be nullified if a deadlock occurs. The call may also be used, with the optional recovery-name specified, to erase previous changes to a shared database (for example if the program detects that a portion of its input has been in error).

Format

ENTER DMSRLSE[,recovery-name]

Usage Rules

The call may be made at any time after all areas are open, but is effectively a null action if no area is opened for shared access, or if no database accesses have been made.

DBM Response

If there are no open areas, the DBM returns an error code in the CCB. If there are areas open, and the optional recovery-name is specified, the DBM restores any before images from the transient journal to the database. If recovery-name is not specified, the DBM writes all modified pages currently in core back to the database. In both cases the DBM:

1. Deletes all before images currently on the transient journal.
2. Sets the program's position in the database to zero; i.e., zeros out all set tables and current-of-type for each group.
3. Releases all locked pages.

Database areas opened to the program are not closed.

Checkpointing

Purpose. To add an additional protection to the integrity of the database by allowing the user's program to periodically request that the DBM write all modified pages to the database.

Format

ENTER DMSCHKPT

Usage Rule

The using program may call the checkpoint routine at any time during its operation.

DBM Response

The DBM will write all modified pages currently in the data buffers to the database area file. After-images will be written to the journal file if journaling is being done. No areas are closed, nor are any currenty indications changed. The database lockout bit will be reset in all updated areas.

Terminating Processing

Purpose. To close opened areas when a program's database activity is finished.

Format

ENTER {CLOSEDB / CLOSAREA, area-name-1 [, area-name-2]...}

Usage Rules

1. CLOSEDB terminates processing in all currently opened areas.
2. CLOSAREA terminates processing of those areas specified by area-name-1, area-name-2, etc.
3. When the last opened area is closed, the user may request that the DBM release back to the monitor any common dynamic core acquired for the subschema and data buffers. The user requests this release of core by setting the contents of cell REF-CODE to a negative value before executing the close call.

DBM Response

The DBM interacts with the host operating system to close the are files. If, however, CLOSAREA is used to terminate processing in an area which has pages enqueued or if the area is open for update and other areas are left open for update then the pages are not released and the operating system close is not issued until the remaining areas are closed with a CLOSAREA or CLOSEDB procedure call.

Error Processing

During execution of an EDMS program, two types of error conditions may occur and be recognized by the DBM. The first type involves data-dependent situations and must be anticipated by the user program. The second type involves situations that result from inproper use of the DBM routine calls, from invalid database definitions reflected in the subschema, from hardware or software malfunctions that cannot be recovered by the DBM, and from deadlock with another program that is sharing an area.

If an error is detected by the DBM, an identifying error code is placed in the ERR-CODE entry of the CCB. If an error-control location was established for the error code encountered, the DBM returns control to that location. If no error-control location exists, control is returned to the location immediately following the DBM function call.

If the error encountered is data-dependent (see Table F-2 in Appendix F), the DBM returns the database to its logical position before the call and makes the appropriate return to the user. Additional DBM calls will be accepted.

If the error is non-data-dependent other than deadlock (see Table F-3), the DBM closes all open areas before returning to the user. If any further calls are made to the DBM, the job is terminated abnormally.

If there is a deadlock, the program's position in the database (i.e., values in the set tables and current-of-type for each group) will be set to zero. The database areas are not closed and subsequent DBM calls will be processed.

Journaling

The DBM includes a facility to optionally create a journal file for each job step that updates an area of the database, thus providing the data necessary to recover the content of the database in the event of hardware or software failure.

The journal file will be generated if an area definition specified journaling, provided the proper DCB assignments are made (see "DBM Operational Interface", below). The journal file is described in Appendix E.

A separate journal, called a transient journal, is created to contain before images for recovery of shared databases. No DCB assignments are needed. The before images on the transient journal contain only the database page image. (See Figure D-1.)

Database Lockout

The DBM will maintain a database lockout bit in page 1 of each area to determine the integrity of the area. If an area is opened for exclusive update, the lockout bit will be set to 1 in the database, just prior to the first write initiated by a user update. The lockout bit is reset to zero when the area is checkpointed or closed by the user. Termination of a program without a user-initiated EDMS close will leave the lockout bit set. If the DBM detects that the lockout bit is set when a user opens an area, an error code is returned to the user in the CCB. The DBM will not set the lockout bit if the area is opened for shared update. It will, however, check if the bit was left set by a previous program.

Summary Statistics Collection

The DDL allows for the specification of statistics collection on group and/or set activity. The DBM will collect the statistics during execution of the user program. These statistics, which are distinct from the run-time statistics described above, provide a historical summary of all jobs affecting the database. The statistics are accumulated in space reserved for them in the user's working storage area and written to a file when the area is closed. The contents of the file may be examined subsequently by means of the Summary Statistics Utility processor (DMSSUMS, see Chapter 5 for a description of this processor). Appendix E shows the format of the statistics file. The statistics collected are

Area-Open Mode, Retrieve, Update, or Create

Total Page Reads and Writes

Total Groups Accessed

Total Groups Inserted

Total Groups Deleted

Group-Total Accesses

Total Inserts

Total Deletes

Set-Total FINDN calls

Total FINDP calls

Total HEAD and FINDM calls

DBM Operational Interface

The DBM will exist either as a nonshared library or as a combination public library and nonshared library at the installation's option. Linking of a user's program to the DBM will depend on the option selected.

Total Nonshared Library

The DBM will exist as three files, :DIC, :LIB, and :BLIB, in account DMSLIB. The files :DIC and :LIB are for use by the overlay loader while :BLIB is used by the one-pass loader.

To link a program to the DBM using the overlay loader, account DMSLIB should be specified as an UNSAT option on the LOAD command. For example,

```
!LOAD (GO),(EF,(SUB1)),(UNSAT,(DMSLIB))...
```

To link a program to the DBM using the one-pass loader, file :BLIB in account DMSLIB should be specified as a library identification in the LINK command. For example,

```
!LINK MYROM ON MYLMN;:BLIB.DMSLIB ...
```

Combination Public and Shared Library

The nonshared portion of the DBM will exist as three files, :DIC, :LIB, and :BLIB, in account DMSLIB. The shared portion will exist as file :Pn, where n is a digit selected at the time the DBM is SYSGENed.

To link a program to the DBM using the overlay loader, account DMSLIB and the file :Pn should be specified as UNSAT options on the LOAD command. For example,

```
!LOAD (GO),(EF,(SUB1)),(UNSAT,(DMSLIB),(:P2))...
```

To link a program to the DBM using the one-pass loader, Pn (the colon is omitted) should be specified as a library search option and file :BLIB, in account DMSLIB, should be specified as a library identification in the LINK command. For example,

```
!LINK (P2) MYROM ON MYLMN;:BLIB.DMSLIB...
```

DBM DCB Requirements

The names for the DCBs used by the DBM are as follows:

Journal DCB – F:JRNL.
Subschema DCB – F:SSCH.

Transient Journal DCB-F:TJRL
Statistics DCB — F:STAT.
Database Area DCBs — F:DBnn, where nn may be any two digits from 01 through 64.

The F:JRNL, F:SSCH, F:TJRL, and F:STAT DCBs are automatically included in the user's load module by the loader. DCBs for the database areas must be included by the user as input to the loader. Element files are included in account DMSLIB for this purpose. The element file names and the DCBs in each file are as follows:

DCB1	F:DB01	1 DCB
DCB2	F:DB02 and F:DB03	2 DCBs
DCB4	F:DB04 through F:DB07	4 DCBs
DCB8	F:DB08 through F:DB15	8 DCBs
DCB16	F:DB16 through F:DB31	16 DCBs
DCB32	F:DB32 through F:DB64	33 DCBs

The user must specify, in the LOAD or LINK command, the proper element file(s) to provide a DCB for each area defined in the subschema used by his program.

Example

Three areas defined in the subschema:

```
!LOAD (GO), (EF, (DCB1, DMSLIB), (DCB2, DMSLIB)), (MAP), (UNSAT, (DMSLIB))
```

The DCBs thus included are F:DB01, F:DB02, and F:DB03. The files for the three areas of the database must be assigned to these three DCBs. It is immaterial which file is assigned fo which DCB.

Example

Four areas defined in the subschema:

```
!LINK MYROM, DCB4. DMSLIB ON MYLMN;:BLIB. DMSLIB
```

The DCBs included are F:DB04, F:DB05, F:DB06 and F:DB07.

DCB Assignments

The database area files and the subschema file may exist in public RAD or disk storage, or on a private disk pack. If they are on a private pack, the appropriate serial numbers must be included in the ASSIGN command. If the files exist in an account other than the one in which the job is to be run, the account-name of the account that owns the files must be specified in the ASSIGN command. A mode is not necessary in the assignment because the DBM will open the files with a mode corresponding to the type of open call initiated by the user for the area.

Example

Subschema and database area named AREA1 on public storage database area, AREA2 on private pack number P124:

```
!ASSIGN F:SSCH, (FILE, MYSUBSCH)
!ASSIGN F:DB02, (FILE, AREA1)
!ASSIGN F:DB03, (FILE, AREA2), (SN, P124)
```

The journal and statistics files may be assigned to a file on RAD or disk storage, or to a labeled tape. A mode is not required because the DBM will default the mode to OUT when the first database area is opened by a program. If the program executes multiple opens and closes of the database areas, the DBM will initiate subsequent opens of the journal and statistics files as INOUT, thus concatenating all of the output for any one job step through these DCBs. If the user wishes to concatenate the output of several job steps, he may assign the DCBs as mode INOUT.

5. EDMS UTILITY PROCESSORS

The utility processors perform a service function in support of the other EDMS capabilities: initializing areas before any data is stored; dumping the total contents of an area and saving it for backup; updating the saved data with journaled pages for recovery purposes; printing selected portions of an area, journal, or backup file for visual checking; and printing summary statistics collected by the DBM into a statistics file.

Database Initialization (DMSINIT)

DMSINIT initializes an area or areas of a database, or specified pages in an area. If a whole area is involved, DMSINIT determines the required size for the area and creates the file by writing page headers and optional checksums on all data and index pages. If inventory is specified in the area definition, DMSINIT writes page headers and optional checksums on the inventory pages and fills in unused space with zeros.

DMSINIT Error Messages are shown in Table F-5, Appendix F.

The user may select the areas to be initialized, or specific pages within selected areas. If no areas are selected, all the areas defined for the database will be initialized. In all cases, the area file must be assigned (see "Utilities Operational Interface", below) if an area is to be wholly or partially initialized. Areas are selected by one or more area statements.

AREA Statements

Purpose. To cause DMSINIT to completely initialize one or more areas, or reinitialize a range of pages within each of one or more areas. A single AREA statement may designate many areas to be completely initialized, but a separate statement is required for each area in which specified pages are to be reinitialized.

Format

$$\underline{\text{AREA}} = \text{area-name-1}\ \left[,\text{area-name-2}[,\text{area-name-3}]\ldots\right.$$
$$\left.\underline{\text{RANGE}}=(r_1,r_2)[,(r_3,r_4)]\ldots\right].$$

Usage Rules

1. The AREA statement must end with a period.
2. At least one space must precede the word RANGE.
3. A space may precede or follow an equals sign, a comma, a left or right parenthesis, or a period.
4. The RANGE option defines the page range or ranges to be initialized for an existing area. Each page range specified is validity-checked to determine that r_1 is equal to or less than r_2, and that the page numbers used fall within the total number of data pages in the area. The RANGE must not include index or inventory pages.
5. Each AREA statement should begin on a new input line, but a statement may be continued on as many lines (records) as are needed. No continuation character is required, as a statement is considered continued until a period is encountered.
6. If the specified RANGE includes any pages within the page range of an indexed group, it must include all pages in that range. The specified RANGE may not include pages within the area's overflow range if it does not include the indexed group's pages, and it must include all pages of the overflow range if it includes any.

Dump Processor (DMSDUMP)

This processor dumps either all or selected parts of existing data base areas to a sequential file or to a printer. When the output is defined as a sequential file, the file has the same format as the journal file except that each data page image is dumped as an after-image. Figures E-1 through E-4 in Appendix E show the journal/dump file format.

When the output from DMSDUMP is defined as printed output and the job is run in batch, each page is formatted as shown in Figure 10. The line indicated by (1) is a print header line containing relative page number and the number of words of available space. The line indicated by (2) contains the two-word page header. The line indicated by (3) contains the decimal representation of the line number of the group occurrence, the group number, the relative position on the page, and the group occurrence's reference code. The line indicated by (4) is the beginning of the actual values in the group occurrence. The line indicated by (5) shows the EBCDIC representation of the data (data that does not convert to printable characters are represented by dots).

When printed output is requested by a terminal job, the output is as shown in Figure 11. The (1) indicates the header line containing page number and number of words of available space. The (2) indicates the two-word EDMS page header (see Figure D-1 for data page header format). The first word of the page header shown in Figure 11 contains page number (1), page type (01, data page), the must-write-flag reset, and the number of words of available space (1EC). The second word contains the Control Set pointer (area 2, page 1, line 2). The printed line in Figure 11, indicated by the (3), contains the line number, group number, relative word position in page, and reference code of the first group occurrence. Group number, printed as zero in this case because page 1 line 1 contains a DBM-generated dummy group occurrence, is in the range 1 to 999 for user-defined groups. The line indicated by the (4) in Figure 11 is the beginning of the actual group occurrence, and the line indicated by (5) is the checksum for the page.

DMSDUMP Error messages are shown in Table F-6 in Appendix F. The processing options of DMSDUMP are selected by input directives consisting of a type identifier followed by one or more area selection specifications.

If the database is password-protected, a password specification must precede the first directive. The password specification has the following form:

PASSWORD = 'user-password'

Should a request be made for a selection of groups whose access codes are not authorized by the password given, the groups will be skipped. Items for which the password is not authorized will be zero filled.

```
(1)
LIN/GPN/WRD/REF-CD     DATA  PAGE  0000005     SPACE AVAILABLE 453
                    (2)000055C5 02000501                                    *
(3)
001/200/002/000501
                    (4)01320007 02000502 000C0014 F2F0F0F0 0000000A 020C0508 02000504      (5)*.............200C.............
002/200/009/000502
                       02320007 02000503 000C002A F2F0F0F0 00000014 02000502 02000502       *.............2000.............
003/200/016/000503
                       03320007 02000500 00000032 F2F0F0F0 00000018 02000503 020C0503       *.............200C.............
004/201/023/000504
                       04324007 00000006 02000501 020005C5 02000504 02000504 00000000       *.. ............................
005/201/030/000505
                       05324007 00000007 02000504 02000506 02000505 02000505 00000000       *.. ............................
006/201/037/000506
                       06324007 00000008 02000505 0200C507 02000506 02000506 00000000       *.. ............................
007/201/044/000507
                       07324007 00000009 02000506 02000508 0200050? 02C00507 00000000       *.. ............................
008/201/051/000508
                       08324007 0000000A 02000507 02000501 02000508 02000508 00000000       *.. ............................
                        CHECKSUM  3A650028
```

Figure 10. DMSDUMP Output Sample (Batch Job)

```
!SET F:SCHE DC/MSTRSCH.

!SET F:DB01 DC/AREA-2

!DMSDUMP.

DMSDUMP - EXTENDED DMS
>PASSWORD='11111111'
>PRINT AREA=AREA-2 CIPHKEY='1234' RANGE=(1,2).

(1) LIN/GPN/WRD/REF-CD      DATA  PAGE  0000001      SPACE AVAILABLE 492

(2)    000015EC 02000102

(3) 001/000/002/000101

(4)    01FA8003 02000308 02000408

002/200/005/000102

   02320007 02000103 00000016 F2F0F0F0 0000000A 02000102 02000102

003/200/012/000103

   03320007 02000100 0000002C F2F0F0F0 00000016 02000103 02000103

(5)                      CHECKSUM  FF40865F

LIN/GPN/WRD/REF-CD      DATA  PAGE  0000002      SPACE AVAILABLE 488

   000025E8 02000201

001/200/002/000201

   01320007 02000202 00000006 F2F0F0F0 00000002 02000201 02000201

002/200/009/000202

   02320007 02000203 00000010 F2F0F0F0 00000008 02000202 02000202

003/200/016/000203

   03320007 02000200 00000026 F2F0F0F0 00000012 02000203 02000203

                      CHECKSUM  F3690D39
```

Figure 11. Sample DMSDUMP Terminal Job

Dump Directives

Purpose. To specify the type of output desired and to identify areas, lines, and groups to be processed. Multiple directives may be supplied. They are processed serially by DMSDUMP in order of input, with no attempt made to minimize passes through the database area.

Format

$$\left\{\begin{matrix}\underline{\text{DUMP}}\\ \underline{\text{PRINT}}\end{matrix}\right\} \left[\begin{array}{l} \underline{\text{AREA}} = \text{area-name} \\ \quad [\underline{\text{CIPHKEY}} = \text{user-cipher-key}] \\ \quad\quad \left[\begin{array}{l} \left\{\begin{matrix}\underline{\text{LINE}}\\ \underline{\text{GROUP}}\end{matrix}\right\} = (N_1, N_2) \\ \quad \underline{\text{GROUP}} = N_3[, N_4]\ldots \end{array}\right] \\ \quad [\underline{\text{RANGE}} = (r_1, r_2)[, (r_3, r_4)]\ldots] \end{array}\right] \ldots .$$

Usage Rules

1. The directive type identifier may begin in any character position and may be followed by any number of spaces, and selection parameters may consist of several lines. A period is used to terminate a directive. At least one space is required to separate two selection parameters. Spaces may precede or follow an equals sign, a comma, a left or right parenthesis, or a period.

2. DUMP/PRINT – Specifies that the selected portion of the database is to be output to a sequential file (DUMP), or to a formatted print report (PRINT). The formatted print report contains the hexadecimal representation with EBCDIC alongside, if the job is run in batch. The output of a terminal job does not include EBCDIC.

3. AREA – Identifies the specific area to be processed. Should AREA not be supplied by the user, all areas of the database will be processed. (Area-name is the name of an existing area to be processed.)

4. CIPHKEY – Specifies that deciphering is required in order to produce the requested print report. (User-cipher-key is the cipher key associated with the data in the area to be printed.)

5. LINE – Specifies the span of lines within a data page to be printed. Not legal if GROUP is specified.

6. GROUP – Is group number, which specifies a span of groups or some specific groups to be printed. Note that CIPHKEY, LINE, and GROUP are not allowed with DUMP and are valid only when the AREA parameter is selected. (N_1, N_2) permits the user to specify a span of lines or a span of group numbers to be processed. $N_3[, N_4]\ldots$ allows the user to specify up to eight group numbers of groups whose occurrences are to be processed. GROUP may not be duplicated for a single area.

7. RANGE – Defines one or more page ranges to be selected from an area of a database. Each range specified is checked to confirm that it falls within the page range of an area (including inventory pages), and the r_1 value is checked to determine that it is equal to or less than the r_2 value. No check is made for overlapping ranges; i.e., all selected pages in each range are output. If no RANGE parameter is supplied, the complete area is selected and sent to the output file. In this case, data, index, and inventory pages are written to the output file. RANGE must be the last parameter specified for an area.

Load Processor (DMSLOAD)

DMSLOAD restores all or selected parts of existing database areas from a sequential file on magnetic tape, RAD, or disk. Its output may be directed to another sequential output or to a printer.

The input file must be a single file created as a journal file by the DBM or a dump file created by the EDMS Dump processor. In either case, the file format is as defined in Appendix E.

When the output is directed to a database area, each page selected is written over (replaces) the corresponding area page. Optionally, the area is reciphered and the inventory pages are updated to reflect the condition of each data page restored.

DMSLOAD must always refer to existing areas of a database. Note that if a specific area no longer exists in the database, the user should initialize it before using DMSLOAD to restore it.

When the output is directed to a sequential file, the selected pages are written to the file in the same sequence and format as they are found on the input file. The ability to write to a second sequential file makes it possible to preselect before- or after-images from a journal file for use in recovering the database.

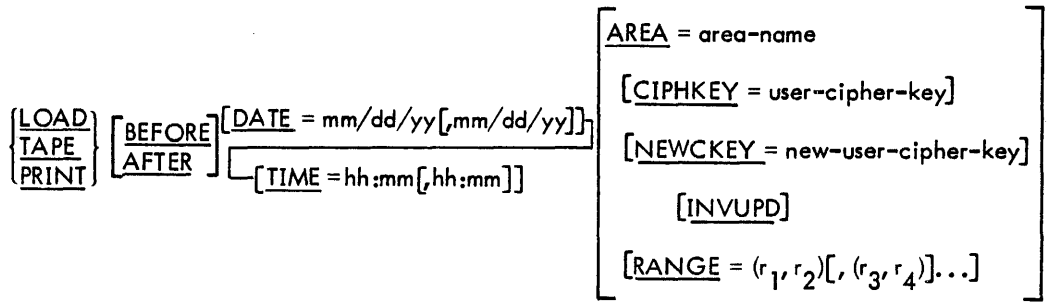
When the output is directed to a printer, the selected pages are formatted the same as in DMSDUMP output (see Figures 9 and 10).

The processing options of the Database Restore routine are driven by directives supplied via the SI input file. A directive consists of a type identifier optionally followed by an AREA selection specification. Each area specification consists of an area identifier optionally followed by one or more area-level selection parameters.

DMSLOAD Directives

Purpose. To specify the form of the output and to select specific types of page images or specific pages to be processed.

Format



Usage Rules

1. Each selection parameter must be separated from the next by at least one space (many spaces are the same as one space). A period is required to terminate a directive. Spaces may precede or follow an equals sign, a comma, a left or a right parenthesis, or a period.

   Each directive must begin on a new input line (record).

2. LOAD – Reloads all or selected parts of an existing database from a sequential file on magnetic tape, RAD, or disk.

3. TAPE – Recreates a sequential file on another magnetic tape, RAD, or disk with its selected output.

4. PRINT – Displays all or parts of the database from a DMS dump tape or journal tape to the printer or terminal.

5. BEFORE or AFTER — Specifies that only the before or after page images are to be selected from the input file. If not specified, both types of page images are selected.

6. DATE and TIME are used to select pages from the input file. When a single date is given, only pages for that date are selected. When two dates are given, an inclusive range is defined and all input pages within that range are selected. Also, the first date must chronologically precede the second. The time parameter is a logical extension of the date parameter and is used in the same manner. If both are used for a given directive, the first time value is assumed to be the time for the first date and the second time value for the second date.

7. AREA – Allows user to specify the area he intends to process. If AREA is not supplied by the user, none of the following area-level selection parameters should appear, and all areas of the database will be processed. (Area-name is the name of an existing area to be processed.)

8. CIPHKEY – Specifies that deciphering is required in order to produce the requested print report in PRINT option or that an area of the database in LOAD option is to be reciphered.

9. NEWCKEY – Specifies that the area defined in the area identifier will be reciphered using a new cipher key. NEWCKEY can only be specified when LOAD directive is selected. (New-user-cipher-key is a one- to four-character string that will be used as a new cipher key to recipher the area specified.)

10. INVUPD – Has meaning only when used with the LOAD directive. When INVUPD is specified, DMSLOAD updates the inventory pages of the area specified with the space available as defined by each page restored. When INVUPD is not specified, it is assumed either that the inventory pages were restored from the tape file by a directive that included the inventory pages or that it is not necessary to update the inventory.

11. RANGE – Selects one or more page ranges within the specified area to be processed. Must be the last paramater specified for an area.

Summary Statistics Processor (DMSSUMS)

DMSSUMS outputs in print format the total contents of the statistics file generated by the DBM or selected counts from that file. The user may select area counts, group counts, or set counts for all or specified area, groups, and sets by means of statistics selection specifications input to DMSSUMS. A valid schema must also be input.

The output from DMSSUMS is in the form shown in Figure 12. The information is output in the order in which it occurs in the statistics file. DMSSUMS error messages are shown in Table F-8, Appendix F. Statistics File format is shown in Appendix E. The statistics file is not modified or deleted by DMSSUMS, it may be extended in subsequent jobs (see "DBM Operational Interface", Chapter 4) or it may be deleted.

```
DMSSUMS   HERE

  DMS SUMMARY STATISTICS
  COLLECTED DURING J00-0000D4  12/26/72 13:50

  AREA-STATISTICS                #GROUP ACCESSES  #GROUP INSERTIONS  #GROUP DELETIONS  #PAGE ACCESSES  OPEN-MODE

  AREA-A                                    216                 27                 7             118  UPDATE

  GROUP-STATISTICS               #GROUP ACCESSES  #GROUP INSERTIONS  #GROUP DELETIONS

   GROUP-A                                   95                 10                 2

  SET-STATISTICS                 FINDN CALLS      FINDP CALLS      HEAD+FINDM CALLS

  SET-A                                     50                                    10

  GROUP-STATISTICS               #GROUP ACCESSES  #GROUP INSERTIONS  #GROUP DELETIONS

  GROUP-B                                   20
```

Figure 12. DMSSUMS Sample Output

Statistics Selection

Purpose. To designate the areas, groups, and sets for which statistics are to be printed.

Format

$$\left[\underline{\text{AREA}} = \left\{\begin{matrix}\text{area-name [, area-name-2]...} \\ \underline{\text{ALL}}\end{matrix}\right\}\right] \ldots$$

$$\left[\underline{\text{GROUP}} = \left\{\begin{matrix}\text{group-name-1 [, group-name-2]...} \\ \underline{\text{ALL}}\end{matrix}\right\}\right] \ldots$$

$$\left[\underline{\text{SET}} = \left\{\begin{matrix}\text{set-name-1 [, set-name-2]...} \\ \underline{\text{ALL}}\end{matrix}\right\}\right] \ldots .$$

Usage Rules

1. The AREA, GROUP, and SET clauses may be input in any order and may span as many input lines as necessary. The period is required to terminate the input. ALL may be specified only once each for AREA, GROUP, and SET.
2. At least one space is required preceding the words AREA, GROUP, and SET, and many spaces are the same as one except that a complete line of spaces is treated as an end-of-file. Spaces may precede or follow the equals sign, the comma, and the period.
3. AREA indicates that statistics for the designated areas are to be printed. The area-names must be in the schema.
4. GROUP indicates that the statistics for the designated groups are to be printed. The group-names must be in the schema.
5. SET indicates that the statistics for the designated sets are to be printed. The set-names must be in the schema.

Utilities Operational Interface

All DMS utilities may be operated in batch mode or from a terminal in CP-V. All four prompt with a > character and treat a line-feed or carriage-return in response to the first prompt as an end-of-file on the input. Input directives and selections are read through the M:SI DCB and print output and error messages are written through M:LO.

DMSINIT

DMSINIT requires file assignments for the schema that describes the areas to be initialized and for the areas themselves. It uses the DCB F:SCHE for the schema and F:DBnn (where nn is any two-digit combination between 01 and 64) for the areas. Any DBnn can be used for any area. If an area is to be updated or dumped by a job run in an account other than the one in which DMSINIT is run, WRITE account should be specified in the area assignment.

Typical Deck Setup Examples (DMSINIT)

1. Initialize all areas of a database:

```
!ASSIGN F:SCHE,(FILE,SCHEMA)
!ASSIGN F:DB01,(FILE,AREA1)
!ASSIGN F:DB02,(FILE,AREA2)
!ASSIGN F:DB03,(FILE,AREA3)
!DMSINIT
```

   No input is supplied because the database contains three areas, all of which are to be initialized.

2. Initialize selected areas of a database:

```
!ASSIGN F:SCHE,(FILE,SCHEMA)
!ASSIGN F:DB01,(FILE,AREA1)
!ASSIGN F:DB03,(FILE,AREA3)
!DMSINIT
 AREA =AREA3,AREA1.
!EOD
```

3. Reinitialize a portion of an existing area:

```
!ASSIGN F:SCHE, (FILE, SCHEMA)
!ASSIGN F:DB01, (FILE, AREA3)
!DMSINIT
 AREA = AREA3  RANGE = (3, 8), (16, 20), (51, 60).
!EOD
```

The result from the above setup is that pages 3 through 8, 16 through 20, and 51 through 60 of AREA3 are reinitialized.

DMSDUMP

The user must supply !ASSIGN cards for the following files used by the Dump processor:

Database schema file (F:SCHE).

Output dump sequential file (required only when Dump directive is used (F:DUMP).

Each area to be processed (F:DBnn).

Typical Deck Setup Examples (DMSDUMP)

1. Dump all areas of the database to a sequential file:

```
!ASSIGN F:SCHE, (FILE, SCHEMA)
!ASSIGN F:DUMP, (LABEL, DUMPDB), (SAVE), ;
!    (SN, 1234)
!ASSIGN F:DB01, (FILE, AREA-1)
!ASSIGN F:DB02, (FILE, AREA-2)
!ASSIGN F:DB03, (FILE, AREA-3)
!DMSDUMP
 DUMP
!EOD
```

The above setup is to dump the database with three areas (AREA-1, AREA-2, and AREA-3) to a sequential file (DUMPDB) on a labeled tape (SN, 1234).

2. Dump a portion of an area of the database to a sequential file:

```
!ASSIGN F:DUMP, (LABEL, DUMPDB), (SAVE), ;
!    (SN, 1234)
!ASSIGN F:DB01, (FILE, AREA-3)
!ASSIGN F:SCHE, (FILE, SCHEMA)
!DMSDUMP
 DUMP     AREA = AREA-3
          RANGE = (51, 80).
!EOD
```

The above shows that the contents of pages 51 through 80 of AREA-3 are dumped to a sequential file on a labeled tape.

3. Output on printer a portion of an area:

```
!ASSIGN F:DB01, (FILE, AREA-2)
!ASSIGN F:SCHE, (FILE, SCHEMA)
!DMSDUMP
 PASSWORD = 'TEST3001'
 PRINT
 AREA=AREA-2          GROUP=16, 30, 101, 298 .
!EOD
```

The result from the above setup is to have all occurrences of group 16, 30, 101, and 298 of AREA-2 printed on printer output. Note that occurrences of groups whose access codes are not authorized by the password will not be printed and zeros will be printed instead of the values of items not authorized.

DMSLOAD

The user must supply !ASSIGN cards for the following files used by the Database Restore processor:

- Input journal or dump file (F:LOAD).

  Database schema file (F:SCHE).

Depending on output functions specified, !ASSIGN cards are required for the following:

Each area (file) of DMS database (F:DBnn).

Output dump tape file (F:DUMP).

Typical Deck Setup Examples (DMSLOAD)

1. Restore database from a dump tape:

```
!ASSIGN F:LOAD, (LABEL, DMSDP), (SN, 1234)
!ASSIGN F:SCHE, (FILE, DMSCHEMA)
!ASSIGN F:DB01, (FILE, AREA-A)
!ASSIGN F:DB02, (FILE, AREA-B)
!ASSIGN F:DB03, (FILE, AREA-C)
!DMSLOAD
 LOAD.
!EOD
```

   The above setup is to restore from labeled tape #1234. Before doing so, user must be sure that AREA-A, AREA-B, and AREA-C exist in the database. (For a nonexisting area, user should initialize one and then use DMSLOAD to restore it.)

2. Display a portion of the database on printer:

```
!ASSIGN F:SCHE, (FILE, DMSCHEMA)
!ASSIGN F:LOAD, (LABEL, DMSDP), (SN, 1234)
!DMSLOAD
 PRINT
  AREA=AREA-B          CIPHKEY='BUG'
   RANGE=(2, 5).
```

   This setup will print pages 2 through 5 of AREA-B from a journal file or dump file.

3. Recover an area using BEFORE images from a journal tape.

```
!ASSIGN F:SCHE, (FILE, DMSCHEMA)
!ASSIGN F:LOAD, (LABEL, JOURNAL), (SN, 1234)
!ASSIGN F:DB01, (FILE, AREA-A)
!DMSLOAD
 LOAD BEFORE AREA=AREA-A.
!EOD
```

   The above setup enables a user to recover AREA-A to its condition prior to the creation of the journal tape.

4. Recipher an area:

```
!ASSIGN F:SCHE, (FILE, DMSCHEMA)
!ASSIGN F:DB01, (FILE, AREA-C)
```

```
!ASSIGN F:LOAD, (LABEL, DMSCHEMA), (SN, 1234)
!DMSLOAD
 LOAD              AREA=AREA-C
  CIPHKEY='BUG'        NEWCKEY='DOGS'.
!EOD
```

This setup changes the cipher-key associated with the area from 'BUG' to 'DOGS'.

DMSSUMS

The user must supply !ASSIGN cards for the following files processed by the summary statistics processor:

The statistics file output by the DBM (F:STAT).

The schema file for the database (F:SCHE).

Typical Deck Setup Example (DMSSUMS)

```
!ASSIGN F:SCHE, (FILE, DMSSCHEMA)
!ASSIGN F:STAT, (LABEL, SUMSTAT), (SN, 5678)
!DMSSUMS
 AREA=ALL GROUP=GROUP-1, GROUP-2, GROUP-3 SET=ALL.
```

This setup causes all area statistics, all set statistics, and the statistics for GROUP-1 and GROUP-2 to be printed from a statistics file on labeled tape.

6. DATABASE ANALYSIS PROCESSOR

The Database Analysis Processor (DMSANLZ) is the portion of the subsystem that performs the restructuring requirements analysis and schedules processes for actual database restructuring. DMSANLZ performs the following major functions:

1. Restructuring Process Control Language (RPCL) processing.
2. Component association and attribute change analysis.
3. Data load sequence determination.
4. Conveyance process generation.
5. Restructuring process and file usage reporting.

By use of the RPCL, the user provides the Restructuring Subsystem with the information needed to access the source and target schemas and database areas. The RPCL also allows the user to specify a preferred sequence for loading the data groups into the target areas and the translation of certain types of items from the source to the target database.

RPCL Syntax

The entry is the major element of RPCL syntax. Each entry is composed of one or more clauses and is terminated by a period, followed by a space. The first clause in an entry identifies that entry. Each succeeding clause begins with a keyword (optionally preceded by a semicolon) identifying the clause. The second and subsequent clauses in an entry may appear in any order, but the syntactic units within each clause must appear in a specified order.

Clauses are composed of words, literals, file identifiers, volume serial numbers, and separators; each of these is considered to be a syntactic unit.

Words

A word is a string of characters that may be either alphabetic or numeric and may contain embedded hyphens. A word must not begin or end with a hyphen and must contain at least one nonnumeric character.

There are two basic types of words, reserved and nonreserved. Reserved words in the RPCL are the same as those in the DDL (see Chapter 3) and are legal only where specifically required. Nonreserved words may be either names or fill words; their usage is context dependent. A given nonreserved word may be considered a fill word at one place in an entry and interpreted as a name at another. For example, the word SEQUENCE in the RPCL pass sequence subentry is considered a fill word if it appears immediately following the keyword PASS, but may also appear as a group name without ambiguity.

Literals

Literals can be either numeric or nonnumeric. A numeric literal is a string of numeric characters having an integer value in the range 1 through 100.

Table 7. Legal Database Attribute Changes (cont.)

DDL Entry/Subentry	DDL Clause/Subclause	Legal Change?
GROUP	NAME	no
	WITHIN	no
	RANGE	yes
	LOCATION MODE	no
	STORAGE	no
	USING	no
	DUPLICATES	no
	NUMBER	no
	PRIVACY LOCK FOR RETRIEVE	yes
	PRIVACY LOCK FOR UPDATE	yes
	STATISTICS	yes
Item	Item-name	no
	PICTURE	no
	TYPE	no
	OCCURS	no
	PRIVACY LOCK FOR RETRIEVE	yes
	PRIVACY LOCK FOR UPDATE	yes
	CHECK	no
INVERT	Inverted-data-item-name	no
	NUMBER	no
	WITHIN	no
	RANGE	yes
	DUPLICATES	no
SET	NAME	no
	OWNER	no
	ORDER	no

Table 7. Legal Database Attribute Changes (cont.)

DDL Entry/Subentry	DDL Clause/Subclause	Legal Change?
SET (cont.)	GROUP-NO (as sort key)	no
	LINKED TO PRIOR	no
	STATISTICS	yes
MEMBER	Member-group-name	no
	INCLUSION	no
	LINKED TO OWNER	no
	SELECTION	no
	ALIAS	no
	ASCENDING/DESCENDING	no
	RANGE KEY	no
	DUPLICATES	no

RPCL Syntax

The entry is the major element of RPCL syntax. Each entry is composed of one or more clauses and is terminated by a period, followed by a space. The first clause in an entry identifies that entry. Each succeeding clause begins with a keyword (optionally preceded by a semicolon) identifying the clause. The second and subsequent clauses in an entry may appear in any order, but the syntactic units within each clause must appear in a specified order.

Clauses are composed of words, literals, file identifiers, volume serial numbers, and separators; each of these is considered to be a syntactic unit.

Words

A word is a string of characters that may be either alphabetic or numeric and may contain embedded hyphens. A word must not begin or end with a hyphen and must contain at least one nonnumeric character.

There are two basic types of words, reserved and nonreserved. Reserved words in the RPCL are the same as those in the DDL (see Chapter 3) and are legal only where specifically required. Nonreserved words may be either names or fill words; their usage is context dependent. A given nonreserved word may be considered a fill word at one place in an entry and interpreted as a name at another. For example, the word SEQUENCE in the RPCL pass sequence subentry is considered a fill word if it appears immediately following the keyword PASS, but may also appear as a group name without ambiguity.

Literals

Literals can be either numeric or nonnumeric. A numeric literal is a string of numeric characters having an integer value in the range 1 through 100.

A nonnumeric literal is a string of characters enclosed in apostrophes. To include an apostrophe in such a literal, two apostrophes must appear in adjacent character positions. The second apostrophe does not become part of the literal. Nonnumeric literals appear only in the PRIVACY EXTRACT clause in the schema entry and the CIPHKEY clause of the area entry. The specific usage determines the maximum size allowed.

File Identifiers

File identifiers are used to provide DMSANLZ with the information needed to access the source and target schemas and database area files. File identifier syntax is described below.

Volume Serial Numbers

Volume serial numbers (VSNs) identify the private disk packs or tape volumes on which schemas, database areas, and DMSDUMP created files reside. Volume serial numbers for devices other than ANS labeled tape may contain from one to four alphabetic or numeric characters. ANS labeled tape serial numbers must always contain six alphabetic, numeric, or blank characters. If blank characters are to be included in ANS labeled tape serial numbers, the volume serial number must be enclosed in apostrophes.

Separators

Separators are required after all syntactic units. They are as follows:

1. The space (blank) is a separator; it must follow all syntactic units in the absence of any other separator. A space may precede or follow any other separator; any number of spaces is equivalent to a single space (except in nonnumeric literals). Every input unit is implicitly followed by a space; thus, no syntactic unit may span multiple input units.

2. The comma is a separator which is legal only where specifically indicated in the syntax; it is never required.

3. The number sign (#) is a separator used exclusively to introduce a volume serial number. It is always required where indicated in the syntax.

4. The semicolon may be used as a separator between clauses; it is never required.

5. The period is a separator when followed by a space or when it is the last character in an input unit. A period is required as the last separator in an entry.

The RPCL is essentially free-form in terms of length of input units (up to 80 characters). The input unit is termed a "line", although the original source input may be from cards, keyboard, or any other character string source. The language has no provision for indicating the continuation of lines; an entry is considered continued until terminated by a period, regardless of the number of lines used. The end of a line, however, terminates a syntactic unit.

The syntax notation used to show the RPCL entry formats is the same as that used in the DDL description (see Chapter 3).

File Identifier Format

A file identifier is used to provide DMSANLZ with the information needed to access a schema or database area file.

Format

```
file name [.[account] [.password]]
```

Usage Rules

1. File-name is the name associated with a monitor file on disk or labeled tape. Account is the account number under which the file was created, if different from the current user's account. Password is the monitor password, if any, associated with the file.
2. The file identifier may not contain any separators, and neither the file name, account, nor password may contain a period.
3. The file identifier may not span multiple input units.

RPCL Entry Formats

The RPCL consists of seven types of entries:

1. Schema entry – one each for the source and target schemas.
2. Area entry – one for each source and target area, as required.
3. Load entry – as required.
4. Override entry – one entry, as required.
5. Refcode entry – one entry for each item, as required.
6. Bypass entry – one entry, as required.
7. End entry – one entry.

The first two RPCL entries must be the schema entries, followed by any required area entries, followed by any required load entries, followed by any override entries, followed by any refcode entries, followed by any bypass entries. The end entry must be last in the RPCL.

Schema Entries

The schema entries supply all information required to access the schema files.

Format

```
{SOURCE}
{TARGET}  SCHEMA IS file-identifier
    [;PRIVACY KEY FOR EXTRACT IS privacy-lock]
    [;DP #vsn [#vsn]...].
```

Usage Rules

1. There must be one schema entry for the source schema and one for the target schema. One of these must be the first entry in the RPCL, followed immediately by the other.
2. The SCHEMA clause must be the first clause in the entry. The file-identifier must identify an existing schema file.
3. The PRIVACY clause supplies the key required to access the schema, if the schema has a PRIVACY LOCK FOR EXTRACT attached to it. The privacy key must exactly match the lock of the schema, or the process will be aborted. The privacy-lock is a nonnumeric literal of up to eight characters. If less than eight characters are supplied, trailing blanks are added to make up an eight-character key.
4. The DP clause provides the volume serial number(s) of the private disk pack(s) on which the schema resides, if the schema is not in public storage.

Area Entries

The area entries supply all information necessary to access database area files, as well as information on how indexed group storage space is to be utilized during loading of the target database area.

Format

```
{SOURCE}
{TARGET}  AREA IS  {area-name DMSDUMP FILE IS dump-file-identifier}
                   {area-file-identifier                          }

[;CIPHKEY IS user-cipher-key]

[;INDEXED GROUP FILL PERCENT IS integer]

                 {DP}
[;{[ANS]  }      {9T}  #vsn [#vsn]...]
  {[ANS]  }      {7T}
```

Usage Rules

1. An area entry is required for any source or target area for which any of the following is true:

   a. The area is a source area and the user wants to use a DMSDUMP file as input to the unload process (see Chapter 7).

   b. The database area file is not in public storage in the run account.

   c. The database area file has a monitor password associated with it.

   d. The area is enciphered.

2. The AREA clause must be the first clause in the entry. If the area is a source area and is to be input from a DMSDUMP file, area-name must be the name of a database area defined in the source schema, and dump-file-identifier must identify an existing DMSDUMP file containing the named area in its entirety. The DMSDUMP option is legal for source areas only. If the area is a target area, or if it is a source area in random file format, area-file-identifier must identify an existing database area file. Also, if the area is a target area, it must exist in an initialized state (i.e., it must not contain any group occurrences).

3. The CIPHKEY clause supplies the enciphering key, if any, associated with the area. The form for user-ciper-key is a nonnumeric literal of up to four characters. If less than four characters are supplied, trailing blanks are added to make up a four-character key.

4. The INDEXED clause allows the user to specify an intermediate fill percent to be used when loading the indexed data group occurrences into the target area. This clause is only required if the user has included groups whose location mode is via or direct in the same page range as the indexed group. In this case, at least two load passes are made over the indexed group's page range. The first pass will load only the indexed group occurrences; the second pass, and any subsequent passes, will load the direct or via group occurrences. The fill percent in the INDEXED clause will be used instead of the fill percent in the target schema DDL in the first pass only. In all subsequent passes, the fill percent in the DDL will be used.

   This technique allows the direct and via details to coreside with the indexed group occurrences in the indexed group's page range. Otherwise, all via and direct member group occurrences would be placed in the overflow range, since the first load pass would have exhausted the available space (up to the percent specified in the target schema DDL) in storing the indexed group occurrences.

   The integer specifying the fill percent must lie in the range 1 through 100, and must be less than or equal to the fill percent specified in the target schema DDL.

   The INDEXED clause is legal for target areas only. If the INDEXED clause is not specified, the fill percent in the target schema DDL is used for all load passes.

5. The volume serial number (vsn) clause specifies the device type (7-track tape, 9-track tape, or private disk pack) and volume serial numbers of the private volumes on which the database area or DMSDUMP file resides. Tape devices are legal only when the area is to be unloaded from a DMSDUMP file. Database area files must always reside on random access devices.

Load Entry

The load entries provide the user with a means of influencing the sequence in which his data will be loaded into the target database (see "Data Loading Sequence", below, for a complete discussion of the data loading sequence and the effects and implications of the use of load entries).

A load entry consists of a load subentry, followed by the preserve subentry, followed by the pass sequence subentries. Load entries are never required.

Load Entry Skeleton

Load subentry

[Preserve subentry]

[Pass sequence subentry]...

The required load subentry identifies the load entry. The preserve subentry and the pass sequence subentries are optional.

Load Subentry

The load subentry identifies the area to be loaded.

Format

LOAD AREA area-name.

Usage Rules

1. Area-name must identify an area defined for the target database.
2. Only one load subentry may be specified for a given area.
3. Wherever possible, target areas will be loaded in the sequence defined by the load subentries.

Preserve Subentry

The preserve subentry provides the user with the capability of directing the Restructuring Subsystem to convey the selected groups from the source database to the target database in such a way as to preserve their physical location.

Format

PRESERVE LOCATION OF {GROUP / GROUPS} {group-name / group-number} [, {group-name / group-number}]... .

Usage Rules

1. The groups identified by group-name or group-number must be defined in the target database as having via or direct location mode, and must reside in the area named in the associated load subentry. Via or direct groups that share their page range with a group whose location mode is indexed, may not be specified in a preserve subentry.
2. If specified, the preserve subentry must immediately follow its associated load subentry, and only one preserve subentry may be specified for a given area.

3. The Restructuring Subsystem is implemented so that the reference codes of all groups specified in preserve subentries will be the same in the target database as they were in the source database, so long as the number of lines per page in the target area is the same as in the source area.

   If the number of lines per page has changed and the source page number and line number are either unavailable or nonexistent, the groups will be placed in the target database as close to their source page and line as possible.

4. The target page range of any group specified in a preserve subentry must be a superset of the source page range.

5. All groups specified in the preserve subentry for an area will be loaded in a special load pass before any other data groups are loaded.

Pass Sequence Subentry

Pass sequence subentries allow the user to define the exact loading sequence of all or any part of the data groups within an area.

Format

PASS SEQUENCE IS $\left\{\begin{matrix}\text{group-name}\\\text{group-number}\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{group-name}\\\text{group-number}\end{matrix}\right\}\right]\ldots$.

Usage Rules

1. The groups identified by group-name or group-number must be defined in the target database as being within the area named in the associated load subentry, and must not have been specified in a preserve subentry or in any preceding pass sequence subentry.

2. Each pass sequence subentry defines a separate load pass, and all groups specified in one pass sequence subentry will be loaded in the same load pass. Within the defined pass, group occurrences will be loaded in the order specified in the pass sequence subentry.

3. The normal loading precedence is: direct groups without storage sets, followed by calc and indexed groups, followed by via groups and direct groups with storage sets. These precedence groupings are given weight factors of 1, 2, and 3, respectively. Note that direct groups with storage sets are conveyed as via groups. Further, each pass sequence subentry is assigned a weight factor equal to that of the lowest-weighted group specified in the subentry. For example, a pass sequence subentry containing a direct group and a calc group would be assigned a weight factor of 1, while one containing a calc group and a via group would have a weight factor of 2. Pass sequence subentries must be sequenced in ascending order by weight factor.

4. No via or direct group may be specified for loading prior to, or in the same pass as, the owner group of its storage set if the storage set owner is in the same area as the via or direct group. Note that this restriction does not apply to the special load pass performed as the result of a preserve subentry.

Override Entry

The override entry allows the user to override the default justification of an item being conveyed from the source to the target database if the size of the item is being increased. The normal justification is left justified for alpahbetic or alphanumeric items and right justified for numeric items.

Format

OVERRIDE DEFAULT JUSTIFICATION OF

data-item-name-1[{OF/IN}group-name-1] [data-item-name-2[{OF/IN}group-name-2]]... .

Usage Rules

1. If data-item-name-n is not unique within the source schema then the optional group-name qualifier must be used.

2. All items defined in the override entry must be one in which the number of characters and or digits has been increased from the source to the target schema.

3. All items defined in the override entry must be one whose internal data format is either EBCDIC or packed decimal and if it is a numeric value it must consist solely of integral digits in both the source and target schema.

4. A change in internal data format between the source and target database is not allowed for any item specified in the override entry.

5. An item may not be specified in both an override entry and a refcode entry.

Refcode Entry

The refcode entry provides for identification of those items in the source database which contain reference code values. The source database value of all refcode item occurrences will be replaced in the target database by its target database reference code equivalent.

Format

REFCODE data-item-name-3[{OF/IN} group-name-3] [SOURCE AREA IS area-name].

Usage Rules

1. If data-item-name-3 is not unique within the source schema then the optional group-name qualifier must be used.

2. If a reference code is defined as a three byte value in the source database, then the item values will be remapped using the area number in which the data group containing the item resides unless the optional source area clause is used. The optional source area clause may be used to specify that the reference code values are for an area other than the one in which the containing data group resides.

3. The internal data type for items specified in the entry must be EBCDIC or binary. An occurs clause for the item is permitted.

4. If an item defined in a refcode entry is increased from three bytes to four bytes then the area number will be added as the higher order byte of each reference code value.

5. An item may not be specified in both an override and refcode entry.

Bypass Entry

The bypass entry allows the user to omit the selection and ordering of nonstorage sets during the restructuring process. For an explanation of the set ordering process see the following subsection 'Selecting Set Occurrences'. The set ordering process may be very time consuming particularly when there are a large number of member occurrences for

a set occurrence and the member occurrences physically lie in disjoint database pages. The benefits of ordering nonstorage sets are the removal of logically deleted member occurrences and the validation of the accuracy of the pointers within a set occurrence. In addition changes may be made to a set description if the set occurrences are ordered.

Format

BYPASS ORDERING OF {SET / SETS} set-name-1[, set-name-2]... .

Usage Rules

1. Sets defined in the bypass entry may not have an optional prior or owner pointer defined in the target schema unless the optional pointer was defined in the source schema.

2. Sets defined in the bypass entry must have all members defined for the set in the target schema that were defined as members in the source schema.

3. Sets defined in the bypass entry must not have the set order changed such that a prior pointer exists in an owner group occurrence of the target database where none existed in the source database (e.g., FIRST to LAST).

4. If a set defined in the bypass entry includes a member whose location mode is INDEXED in the target database then a warning will be issued by DMSANLZ. If such a set has any logically deleted member occurrences in the source database then the unload step of the DMSREST process will terminate with an error message and the restructuring process will not be carried out.

5. Sets defined in the bypass entry must not be defined as the VIA or STORAGE set for a group in the target database.

End Entry

The last entry in the RPCL must be the end entry.

Format

END.

The user should be aware that there is no capability in the Restructuring Subsystem for introduction of new data into the target database. Thus, if a new area is defined, then that area will not be accessed by DMSREST unless existing data groups have been specified as being resident in the new area. If a new data group has been defined, there will be no occurrences of that group in the target database. If data items are added to an existing group then the item occurrences will be set to appropriate null value, i.e., zero for numeric and space for alphanumeric. If an existing data item has an occurs clause added or increased then the n values from a source database item will be placed in the first n values of the target database item and the new value occurrences set to a null value. If an existing data group is defined as a member of a new set then the group occurrences will be designated as unlinked in that set. If an existing data group is defined as the owner of a new set, then all group occurrences will be designated as the owner of an empty set. If the order of an existing set is changed (e.g., NEXT to LAST) the order of the member occurrences in the target database is not changed by DMSREST.

Data Loading Sequence

The single most important factor in database restructuring is the data loading sequence. The order in which group occurrences are loaded into the target database can significantly affect performance, especially with respect to page overflows. An example of this would be the case where the user has two invert groups sharing a common page range. The first of these (group A) is used primarily as an alternate retrieval path to the parent group (e.g., the parent is retrieved by the sequence: FINDX, A ; FINDD). The second invert group (group B) is used as the object of a FINDSEQ call. If all occurrences of group B are loaded first, the probability that an occurrence of group A will overflow its assigned base page is significantly increased. This can severely slow down the FINDX, since there is a potential for multiple I/O operations to retrieve a single occurrence of group A that has overflowed its base page. Conversely, if occurrences of group A are loaded first, the overflow probability is greatly reduced. Since occurrences of group B are seldom, if ever, retrieved directly, it is of little consequence whether they overflow, at least from the standpoint of retrieval.

Default Data Loading Sequence

The creation of the data loading sequence can be thought of as a two-step process. In the first step, the sequence is established within each area; in the second, the order for processing the areas is determined.

In terms of the data loading sequence, there are three categories of groups. Each group is categorized according to its location mode, and each category is assigned a weight factor, as follows:

Weight Factor	Category
1	Direct groups for which no storage set is specified.
2	Calc groups and indexed groups.
3	Via groups and direct groups for which a storage set is specified.

In the default case (i.e., where no RPCL preserve or pass sequence subentries are specified) groups are organized into load passes by category. Each load pass is assigned a weight factor equal to that of the category being loaded, and load passes are scheduled for execution in ascending order by weight factor. There will be one load pass each for categories 1 and 2, and one for each level of category 3 groups. It is important to note that level determination is based on storage sets only, and that no group with a storage set will be loaded by default prior to, or in the same pass as, the owner group of the storage set. Note also that groups whose location mode is via are stored relative to their via set, unless otherwise specified in the DDL. Example 3 shows a sample default loading sequence.

In cases where consecutive default load passes contain groups of different location modes, some optimization may occur when there is no page range overlap. Example 4 shows how such optimization may occur; Example 5 illustrates a case of partial optimization.

Component Association and Attribute Change Analysis

In the schema DDL, every database component (i.e., area, group, item, and set) is assigned a name that is either unique in itself or is capable of being uniquely referenced (e.g., by qualification). DMSANLZ uses this unique name to associate corresponding components in the source and target schemas. If a component with a specific name exists in the source schema but not in the target schema then that component is considered as deleted from the source database. If a component with a specific name exists in the target schema but not in the source schema then that component is considered as added to the target database.

There are very few restrictions placed on the changes made to the logical structure of the source database. Bear in mind, however, that changes made to the structure may require changes to existing programs and subschemas generated to access that structure. The structure changes which are not allowed are as follows:

- The number assigned to an area may not be changed between the source and target schemas.
- The number assigned to a data group or invert item may not be changed between the source and target schemas.
- The location mode of a data group may not be changed to INDEXED in the target schema unless it was defined as CALC DUPLICATES NOT ALLOWED in the source schema.
- A group defined with location mode of INDEXED may not have it's indexed control items changed, replaced, or deleted so that duplicate keys may exist in the target database where they did not in the source database.
- The target database page range specified for a group with DIRECT location mode or one specified in an RPCL preserve entry must be a superset of the source database page range.
- A member of a set defined with MANUAL or OPTIONAL AUTOMATIC inclusion in the source schema may not be changed to AUTOMATIC inclusion in the target schema.
- If a data group which exists in the source schema is defined as a member of a new set in the target schema then its inclusion in that set must be MANUAL or OPTIONAL AUTOMATIC.
- The order of a set may not be changed to SORTED between the source and target SCHEMAS.
- A set which exists in the source and target schemas with order SORTED may not have the sort control items for the members of the set changed so as to potentially alter the order of the member occurrences in a set occurrence.
- The internal data type for a data item may not be changed as follows:

  Non-integer numeric to alphanumeric.

  Alphabetic to signed numeric, numeric, binary, floating point, or packed decimal.

  Alphanumeric to signed numeric, binary, floating point, or packed decimal.

  Signed numeric to alphabetic or alphanumeric.

  Numeric to alphabetic.

  Binary to alphabetic or alphanumeric.

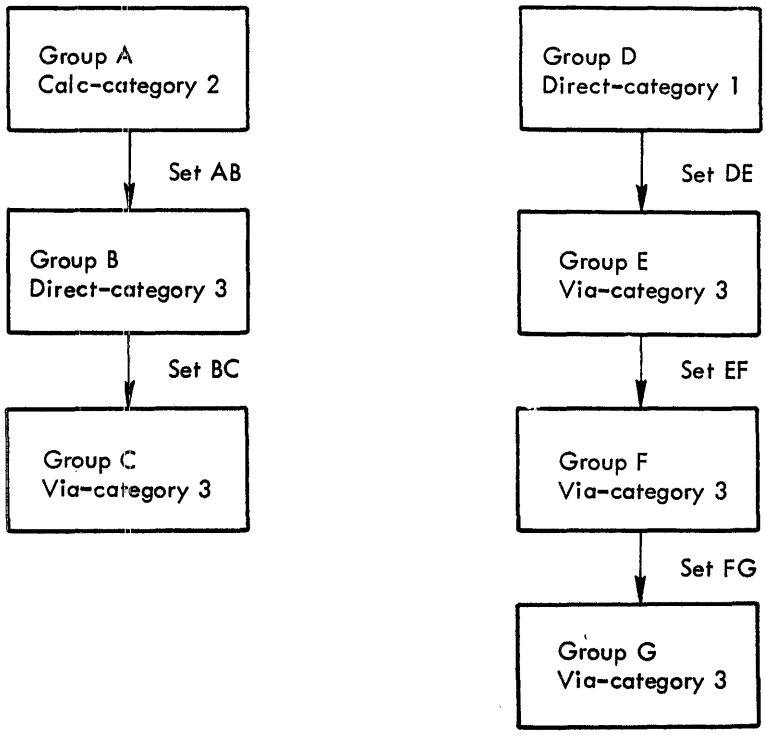
  Floating point to alphabetic or alphanumeric.

  Packed decimal to alphabetic or alphanumeric.

All differences in the description of the source and target database are reported by DMSANLZ. Any such differences that are within the scope of the capabilities of the Restructuring Subsystem are reported as warnings; all other differences are reported as errors.

Example 3. Default Loading Sequence

Consider the following structure for a single area. All sets shown are storage sets. Note that group B is in category 3 (even though its location mode is direct) since it has a storage set.



Assuming that all groups have overlapping page ranges, the default load sequence would be:

Pass	Groups Loaded
1	D
2	A
3	B, E
4	C, F
5	G

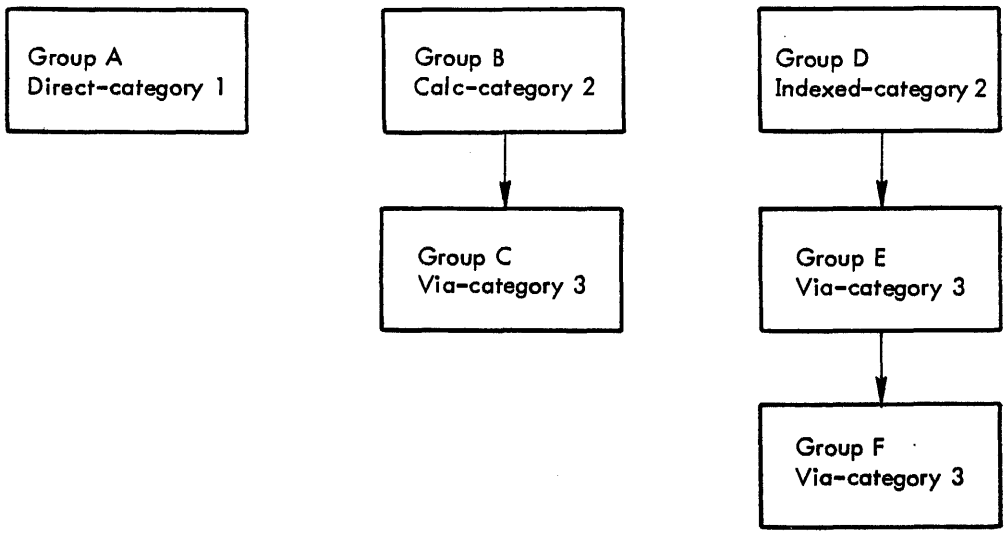
Example 4. Loading Sequence with Optimization

Consider the structure of Example 3, and assume that groups D and A have disjoint page ranges. Optimization would occur as follows:

Pass	Groups Loaded
1	D, A
2	B, E
3	C, F
4	G

Example 5. Loading Sequence with Partial Optimization

When an entire load pass cannot be optimized, partial optimization may still occur. Consider the following single-area structure:



The unoptimized loading sequence would be:

Pass	Groups Loaded
1	A
2	B, D
3	C, E
4	F

However, since groups A and D have disjoint page ranges, by definition, DMSANLZ would include group D in the first pass with group A, as follows:

Pass	Groups Loaded
1	A, D
2	B
3	C, E
4	F

In this case, even though the optimization occurred, there was no saving with regard to the number of load passes. If, however, groups B and E also had disjoint page ranges, one pass could have been eliminated:

Pass	Groups Loaded
1	A, D
2	B, E
3	C, F

Here, two partial optimizations occurred, resulting in a saving of one load pass.

DMSREST loads group occurrences into the target database by first assigning a base page to each group occurrence in the load pass. The base page assigned is dependent upon the group's location mode. The assignment is accomplished as follows:

Location Mode	Assignment
Direct without storage set	The source database page is assigned as the base page in the target database. The group is placed at the same line in the target database that it occupied in the source database if that line is available.
Calc	The calc control items are hashed (using the DBM hashing algorithm) to determine the base page.
Indexed	The group occurrences are sorted in ascending sequence on the index control items. A base page is then determined by the loading process depending upon the space available in the indexed group's page range and any fill percent specified in the RPCL or target schema DDL.
Via and Direct with storage set	If a preserve subentry was specified, the source database page containing the group occurrence is assigned as the target base page. If there was no preserve subentry for the group, then the base page for a group occurrence is the page in the target database that contains the owner group of the storage set occurrence of which the group occurrence is a member.

After a base page is assigned, group occurrences are loaded in a pass ordered on base page and pass sequence number for the group. If the base page does not contain space for a group occurrence, the loading process will search forward in the area for available space. This search is constrained by any page range specified for the group in the target schema.

When more than one group is loaded in a single pass, the order in which the groups are stored into pages may be a factor affecting the efficiency of the target database. This situation is somewhat similar to that described earlier, wherein occurrences of the first group loaded into a page are less likely to overflow that page than are occurrences of subsequent groups. There is an inherent difference, however, in the net result of loading two groups in separate passes versus loading them in the same sequence but in a single pass. Example 6 illustrates this difference. In default load passes, groups are ordered in their target schema DDL sequence. Thus, if occurrences of groups A and B are being loaded into the same page in the same load pass, all occurrences of group A will be loaded first, followed by all occurrences of group B, if group A precedes group B in the target schema DDL.

In the default case, the loading sequence for areas is the order in which they are specified in the target schema DDL. Once an area is scheduled for loading, all groups within that area will be loaded before switching to the next area, provided the storage set owners for any category 3 groups reside in the current area or in another previously loaded area. Example 7 shows the loading sequence for a multi-area structure containing a storage set that crosses area boundaries.

User Influenced Data Loading Sequence

The user can influence the data loading sequence by means of the RPCL load entry. It is strongly recommended, however, that the user run DMSANLZ first, without any RPCL load entries, and examine the generated default loading sequence. If the default sequence does not meet the user's requirements, he can include whatever load entries are needed.

The easiest way for the user to influence the loading sequence is with the preserve subentry. All groups specified in a preserve subentry are designated for loading in the first load pass for the area with which the preserve subentry is associated. The reference codes of all groups specified in a preserve subentry are guaranteed to be the same in the target database as in the source database, provided the number of lines per page in the target area has not changed from its source value, and that all source database pages also exist in the target database. If the number of lines per page has changed, groups will be stored at the same page number and line number whenever possible, or at the closest available location. Note that the only way the user can have a category 3 group loaded before the owner group of its storage set, is to specify the category 3 group in a preserve subentry. Example 8 shows how the preserve subentry could be used to preserve the location of all occurrences of a group.

The user can also influence the data loading sequence by use of RPCL pass sequence subentries. Each pass sequence subentry defines one load pass. DMSANLZ will not try to optimize load passes by adding groups to a user-defined load pass (i.e., one defined by a preserve or pass sequence subentry). Any groups not specified in pass sequence subentries are handled by the default mechanism described earlier.

Example 6. Loading Two Groups in a Single Pass

Assume that groups A and B share a common page range and that they are being loaded in the same pass. The following illustrates the result when page overflow occurs on page n.

Data Page n

Occurrences of group A assigned to page n
Occurrences of group B assigned to page n

Data Page n + 1

Overflow of group B occurrences assigned to page n
Occurrences of group A assigned to page n + 1

Data Page n + 2

Overflow of group A occurrences assigned to page n + 1
Overflow of all group B occurrences assigned to page n + 1
Occurrences of group A assigned to page n + 2
Occurrences of group B assigned to page n + 2

Note the overflow of group A occurrences from page n + 1 into page n + 2, and contrast this result with the following case illustrating a similar situation, where group A is loaded in a separate pass before group B is loaded.

Data Page n

Occurrences of group A assigned to page n
Occurrences of group B assigned to page n

Data Page n + 1

Occurrences of group A assigned to page n + 1
Overflow of group B occurrences assigned to page n

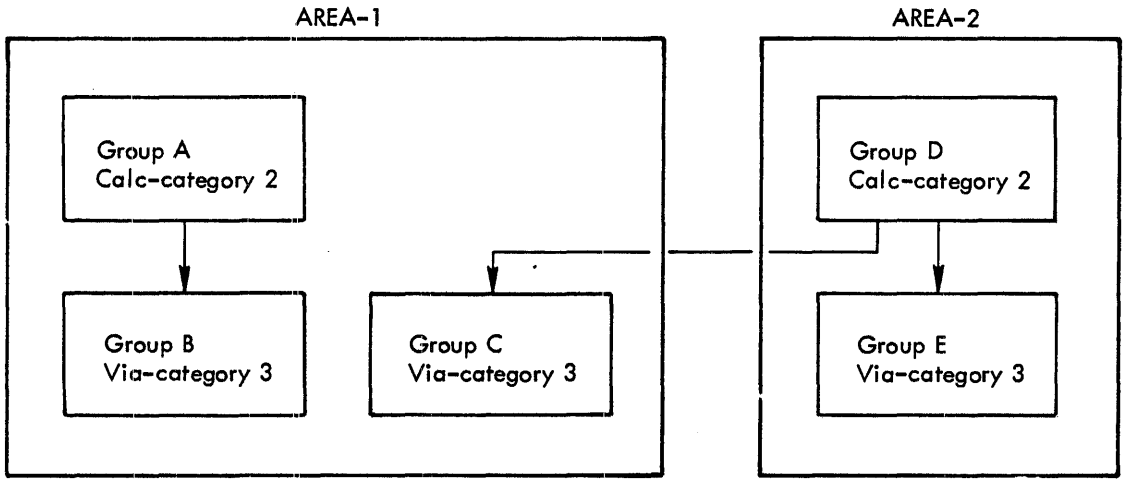
Data Page n + 2

Occurrences of group A assigned to page n + 2
Overflow of group B occurrences assigned to page n
Overflow of all group B occurrences assigned to page n + 1
Occurrences of group B assigned to page n + 2

In this case overflow of group A is eliminated at the expense of allowing group B occurrences assigned to page n to overflow two pages. Note that in the first example, even though group A occurrences overflowed page n + 1, no occurrences of either groups A or B overflowed to more than one page.

Example 7. Loading Sequence for a Multi-Area Structure

Consider the following structure; where all sets shown are storage sets.



The default loading sequence would be as follows:

Pass	Groups Loaded	Area
1	A	AREA-1
2	B	AREA-1
3	D	AREA-2
4	E	AREA-2
5	C	AREA-1

Example 8. Loading Sequence Using Preserve Subentry

Consider the structure of Example 3 and assume that a requirement exists to preserve the location of all occurrences of group 3. If only group B is specified in a preserve subentry, the loading sequence would be:

Pass	Groups Loaded
1	B
2	D
3	A
4	C, E
5	F
6	G

Note that even though there is no apparent potential for conflict should groups B and D be loaded in the same pass, DMSANLZ will not attempt to optimize load passes. If the user desires to have groups B and D loaded in the same pass he must specify both B and D in the preserve subentry.

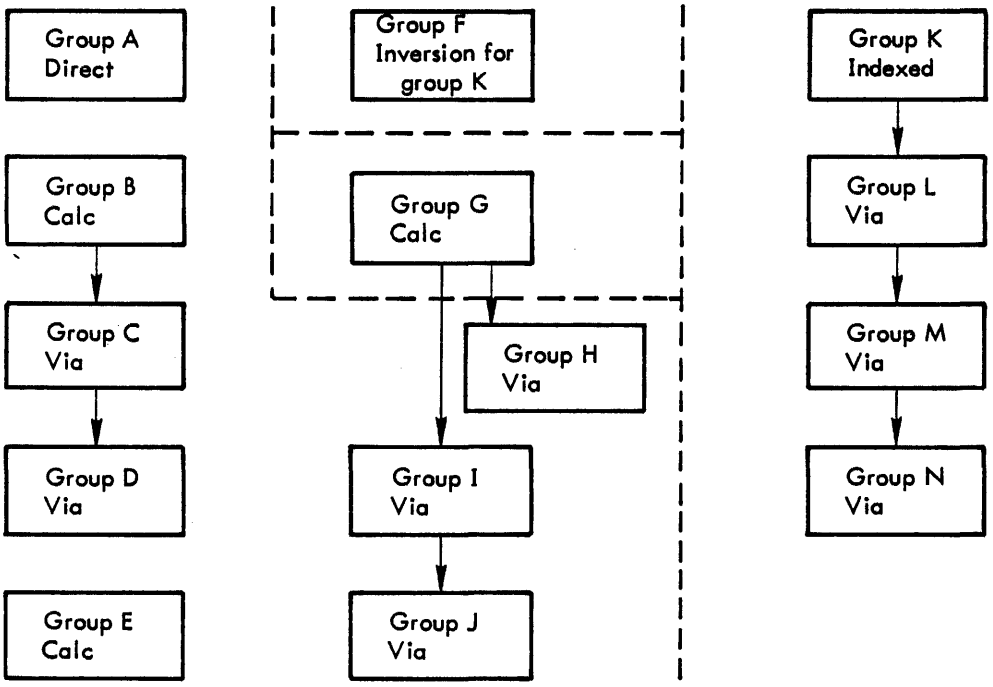
Each user-defined load pass is assigned a weight factor in a manner similar to that of default load passes. Since user-defined load passes may be specified to contain groups from different categories, the weight factor of a user-defined load pass is equal to that of the lowest weighted group in that load pass. User-defined load passes are scheduled for execution immediately preceding default load passes of the same weight. Where multiple user-defined load passes have the same weight, they are executed in the order in which they are defined in the RPCL. Examples 9 and 10 illustrate how pass sequence subentries could be used to influence the order in which groups are loaded.

The user can also influence the area loading sequence by the order of his RPCL load entries. In the default case, areas are scheduled for loading in the order in which they are specified in the target schema DDL. If load entries are specified, the indicated areas are scheduled to be loaded first (in the order specified) and any areas for which no load entry is supplied are scheduled last, in their DDL order.

Once an area is scheduled for loading, all groups within that area are loaded before any groups are loaded into the next area, provided the storage set owners for any category 3 groups reside in the current area or in another previously loaded area. Example 11 shows how a load subentry could be used to improve upon the default loading sequence.

Example 9. Loading Sequence Using Pass Sequence Subentry, I

Consider the following single-area structure, bearing in mind that only storage sets are shown. The area is logically divided into four mutually exclusive page ranges, depicted by the dotted lines.



The default loading sequence for this area would be:

Pass	Groups Loaded
1	A, F, G, K
2	B, E, L
3	C, H, I, M
4	D, J, N

If the user had a specific requirement to load group B before group E, he might include the following load entry in his RPCL:

```
LOAD SAMPLE-AREA
PASS SEQUENCE IS B.
```

This would generate the loading sequence:

Pass	Groups Loaded
1	A
2	B
3	E, F, G, K
4	C, H, I, L
5	D, J, M
6	N

Notice that since group B was specified in a separate load pass, groups F, G, and K were not included in the first pass with group A. This resulted in two extra load passes. One of these extra passes could be avoided by specifying groups F, G, and K to be loaded in the same pass with B, resulting in the loading sequence:

Pass	Groups Loaded
1	A
2	B, F, G, K
3	E, L
4	C, H, I, M
5	D, J, N

Still another pass could be eliminated if the user allowed groups C, H, and I to be loaded in the same pass as group E. This alternative, however, may pose a problem with respect to page overflow, as shown in Example 6. The user should carefully analyze the activity level of group E, and the usage frequency of all retrieval paths by which group E is accessed before making this decision. Also, groups K, L, and M would have to be loaded in the first, second, and third passes, respectively. This poses no problem, however, since the page range for the K-L-M-N substructure is exclusive. The following RPCL load entry would provide this optimized loading sequence:

```
LOAD AREA SAMPLE-AREA.
PASS SEQUENCE IS A, K.
PASS SEQUENCE IS B, F, G, L.
PASS SEQUENCE IS E, C, H, I, M.
```

The generated loading sequence would be:

Pass	Groups Loaded
1	A, K
2	B, F, G, L
3	E, C, H, I, M
4	D, J, N

Example 10. Loading Sequence Using Pass Sequence Subentry, II

Consider the structure of Example 9 and assume that groups I and J are to be loaded before group H. This could be accomplished with the following RPCL load entry:

```
LOAD SAMPLE-AREA.
PASS SEQUENCE IS I.
PASS SEQUENCE IS J.
```

Seven load passes would be generated:

Pass	Groups Loaded
1	A, F, G, K
2	B, E
3	I
4	J
5	C, H, L
6	D, M
7	N

There are several ways the user could optimize this loading sequence. One pass could readily be eliminated, with no side effects, by loading group L in the fourth pass with group J. Groups M and N would then be loaded in the fifth and sixth passes, respectively, eliminating the need for a seventh pass.

Another possibility is to load groups B and E in the first pass. This, however, might cause the reference codes of occurrences of group A to change. For example, if three occurrences of groups B and E overflow from data page n into data page n + 1, they will be assigned line numbers one, two, and three. If a group A occurrence assigned to page n + 1, line two, is then encountered, it will be reassigned to the first available line number; in this case, line number four.

If group A were a single-occurrence header group residing in page one, loading groups B and E in the same pass as group A would not cause a problem, provided group A precedes groups B and E in the group sequence subentry. The following RPCL load entry would cause five load passes to be generated:

```
LOAD SAMPLE-AREA.
PASS SEQUENCE IS A, B, E, F, G, K.
PASS SEQUENCE IS I.
PASS SEQUENCE IS J, L.
```

The generated loading sequence would be:

Pass	Group Loaded
1	A, B, E, F, G, K
2	I
3	J, L
4	C, H, M
5	D, N

One additional pass can be eliminated if the user is willing to let group C be loaded in the same pass as group J. Again, this decision should take into account activity levels and retrieval strategies. Also groups L, M, and N would have to be loaded in the second, third, and fourth passes, respectively.

```
LOAD AREA SAMPLE-AREA.
PASS SEQUENCE IS A, B, E, F, G, K.
PASS SEQUENCE IS I, L.
PASS SEQUENCE IS J, C, M.
```

The following loading sequence would be generated:

Pass	Groups Loaded
1	A, B, E, F, G, K
2	I, L
3	J, C, M
4	D, H, N

Example 11. Load Subentry Used to Improve Default Loading Sequence

Consider the structure of Example 7. The user could improve upon the default loading sequence by including a load subentry for AREA-2. The generated loading sequence would be:

Pass	Groups Loaded	Area
1	D	AREA-2
2	E	AREA-2
3	A	AREA-1
4	B, C	AREA-1

Conveyance Process Generation

DMSANLZ determines the processes that must be performed and the sequence of execution required to accurately convey the user's data from his source database to his target database. This determination is based solely upon the structure of the source and target databases; no consideration is made regarding the volume of data to be conveyed.

The major functions required to restructure an EDMS database are:

1. Unloading of all data group occurrences from the source database.
2. Identification of the storage set occurrences for all via groups and direct groups with storage sets.
3. Loading of all data group occurrences into the target database.
4. Relinking of all set occurrences in the target database.
5. Reporting of any errors found.

In performing these functions, the Restructuring Subsystem makes use of 11 separate processes (see Table 8). Typically, each of these processes is executed a number of times during the course of restructuring a database. The unload process, for example, is executed once for each source database area. Each process execution is termed a "step". When he is running DMSREST, the user controls step execution to the extent that he may specify the number of restructuring steps to be executed in a single CP-V job step. The user may also stop DMSREST at the end of any step (see Chapter 7).

Internal File Handling

The data conveyance process potentially involves the creation of a large number of intermediate files. To facilitate the identification and handling of these files, DMSANLZ names each scheduled intermediate file. These names are composed of an Internal File Descriptor (IFID) followed by the number of the DMSREST step which will create the file. When a file is created, a time suffix is optionally appended to the intermediate name. All intermediate files created and used by DMSREST are thus cataloged in the RPCC and cataloged by the CP-V file system using a name consisting of "IFID – step number [– time]".

Table 8. Restructuring Processes

Process	Mnemonic	Description
Unload	UNLD	Unload group occurrences from one source area.
Polyphase set recovery	PPSR	Recover all sets whose owner resides in a given source area.
Set extraction	XSET	Extract and validate occurrences of sets recovered in one set recovery step.
Assign source base page	ASBP	Identify the owner group occurrences for all via groups residing in one source area.
Assign target base page	ATBP	Assign a target database page number to all via groups for one load pass.
Load	LOAD	Make one load pass across a target database area.
Link	LINK	Relink all set occurrences in one target database area.
Update	UDAT	Update an intermediate file with data from another intermediate file.
Sort all	SRTA	Sort an entire intermediate file.
Sort select	SRTS	Sort selected records from an intermediate file.
Wrap up	WRAP	List all detected conveyance errors and generate the reference code correspondence file.

An IFID indicates (to some extent) the format of a file and the nature of the data contained in the file. Many intermediate files may have the same IFID (but different step numbers), and some potential IFIDs may never be needed for a given database. Table 9 lists the IFIDs that may be scheduled by the Restructuring Subsystem.

When a given process is scheduled for execution, only those files that could be generated by that process are scheduled for creation. The unload process, for example, has the capability of creating a CIDG file. The CIDG file, however, is scheduled for creation only if the area to be unloaded is defined as containing an indexed data group. In the case where such a group is defined, but no occurrences of that group exist, the CIDG file is not created even though it is scheduled for potential creation. This convention of not physically creating null files is followed throughout the Restructuring Subsystem.

When DMSREST is preparing to execute a process, it first determines which input files that process may require. If all input files scheduled for a given process were not created, that process is not executed. Thus, if the source database schema defines components for which no occurrences exist, DMSANLZ may schedule files for creation that in fact are not created, and may schedule processes that will not be executed. Appendix I illustrates the flow of scheduled files through scheduled processes.

Unloading the Source Database

The first major operation required to restructure an EDMS database is the unloading of all data group occurrences from the source database. DMSANLZ schedules one unload step for each source database area.

After all of the unload steps, DMSANLZ schedules one sort for each potential CIDG file. These sorts order the indexed data groups by their index key values in preparation for loading them into the target database.

Table 9. Internal File Descriptors (IFIDs)

IFID	Mnemonic	Contents
Conveyed calc and direct data groups	CCDG	Calc and direct group images.
Conveyed indexed data groups	CIDG	Indexed group images.
Conveyed via data groups	CVDG	Via group images.
Conveyed target data groups	CTDG	Group images for one load pass.
Conveyed reference code occurrences	CRCO	Set pointer occurrences.
Conveyed set key data	CSKD	Set pointer occurrences.
Conveyed sets topologically ordered	CSTO	Set pointer occurrences.
Conveyed group occurrences reference code	CGOR	Reference codes of all conveyed data group occurrences.
Conveyed error group reference codes	CEGR	Reference codes of group occurrences causing errors.
Conveyed storage master reference codes	CSMR	Reference codes of storage set owner occurrences.
Conveyed group's reference codes	CGRC	Corresponding source and target database reference codes for conveyed group occurrences.
Source database area	SDBA	User's source database.
Target database area	TDBA	User's target database.
Set recovery work files	WRK1 ⋮ WRK8	Scratch files for set recovery.

Selecting Set Occurrences

To ensure that occurrences of via groups and direct groups with storage sets are loaded into the target database properly, it is necessary to identify the storage set owner occurrence associated with each such via or direct group occurrence. This entails the "recovery" of every occurrence of each storage set, since head pointers may not be present in some or all of the member group occurrences. Set recovery consists of ordering a file of set pointers so that each set occurrence appears as an owner group occurrence physically followed by all of its member group occurrences.

A significant by-product of set recovery is the ability to validate set occurrences and to detect any inconsistencies in the set pointers. For example, if group occurrence A contains a next pointer for some set pointing to group occurrence B, and group occurrence B contains a prior pointer for that same set that does not point back to group occurrence A, something is obviously wrong. The set recovery procedure detects and reports any such irregularities regarding set pointers. Also, although set recovery is required only for storage sets, recovery is performed for nonstorage sets in order to validate these sets for logical consistency.

The major processes involved in selecting set occurrences are: set recovery, set extraction, sorting, and source base page assignment. The number of times each of these processes is executed depends upon the source database structure. The basic pattern in which DMSANLZ schedules these processes is:

1. Recover and validate the storage sets for each source area. One set recovery process and one set extraction process are scheduled for each source area containing a group defined as owner of a storage set.

2. Identify the storage set owner occurrences for all via group occurrences and for all direct group occurrences that have storage sets. Two sort processes and one source base page assignment process are scheduled for each source area containing via groups or direct groups with storage sets.

3. Recover and validate the nonstorage sets for each area. One set recovery process and one set extraction process are scheduled for each source area containing a group defined as the owner of a nonstorage set.

Loading the Target Database

The main factor influencing the loading of the target database is the data loading sequence, described earlier in this chapter. One sort process and one load process are scheduled for each load pass. If via groups or direct groups with storage sets are to be loaded, however, DMSANLZ schedules another sort process and a target base page assignment process prior to the usual sort and load.

Relinking Set Occurrences

After all required load passes have been scheduled, DMSANLZ schedules the processes needed to map source reference code items and set pointers to their target values, and to relink the set occurrences in the target database. These processes include five sorts and two updates, followed by one relink process for each area.

Error Reporting

The last processes scheduled by DMSANLZ will include two sorts, one update, and a reporting process. These will produce a listing of all errors encountered during DMSREST execution and an equivalence file of source and target database reference codes.

DMSANLZ Reports

DMSANLZ produces three reports that provide the information necessary to interface with DMSREST. These are the

1. Data Load Sequence Listing

2. Scheduled Process Sequence Listing

3. Scheduled Files Listing

Appendix J contains samples of these reports.

Data Load Sequence Listing

This report details the sequence in which groups will be loaded into the target database. It itemizes the sequential load pass number, the DMSREST step number scheduled to execute the load pass, the area that will be loaded, and the names of all groups being loaded. The groups are listed in the order in which they will be loaded in each pass.

Scheduled Process Sequence Listing

All DMSREST processes scheduled for execution are listed in their order of execution. The names of all groups or sets scheduled for use by a process are listed for that process. If the scheduled process is an unload, the group names are listed following the IFID assigned to the file on which the groups will be written. This information may be used to determine the volume of data that will be output through an IFID, and thus any resource assignments that may be desirable for execution of DMSREST. In addition, all files used by a process are listed for that process.

Scheduled File Listing

All files scheduled for use by DMSREST are listed in their order of creation. The default device type, a cipher indicator, logical record length, and any volume serial numbers are listed for each file. In addition, the report will contain the DMSREST step number of the process creating the file, as well as those that will read the file.

!DMSANLZ Control Command

The batch monitor control command to execute DMSANLZ has the form:

```
!DMSANLZ [option][,...]
```

The corresponding on-line command is:

```
!DMSANLZ
```

DMSANLZ prompts the on-line user with a colon (:) for any options. A carriage return or line feed is interpreted as indicating the end of the list of options.

Options may be specified in any order; no options, however, may be repeated. Table 10 lists the options and their meanings. The control command and a listing of the specified and default options is output via the M:LO DCB. Any errors in the !DMSANLZ command are listed via the M:DO DCB; the RPCC is not created, and DMSANLZ terminates. Table 11 lists the diagnostic messages for option errors.

Table 10. Control Command Options

Option	Meaning
LOG NOLOG	Record in the RPCC that all processes executed in DMSREST shall log a start message on the operator's console. Alternatively, record in the RPCC that logging shall not be performed. The DMSANLZ default is LOG.
WORK = w	Record in the RPCC that all set recoveries to be executed by DMSREST shall employ w work file DCBs, where $3 \leq w \leq 8$. The DMSANLZ default is 8.
DMS	Record in the RPCC that the database being restructured is a Basic versus Extended DMS database. This option is required if the database is a Basic DMS database.

Table 10. Control Command Options (cont.)

Option	Meaning
SORT = s	Record in the RPCC that all sorts to be executed by DMSREST shall use s work file DCBs, where $3 \leq s \leq 17$ The DMSANLZ default is 17.
BUFFERS = $\left\{\begin{matrix}1\\2\end{matrix}\right\}$	Record in the RPCC that either one or two buffers may be allocated by DMSREST to each IFID intermediate file, and the source and target database areas. The DMSANLZ default is 2.
BLOCK = k	Record in the RPCC that all IFID intermediate files to be created by DMSREST shall have a nominal blocksize of k bytes, where 8 + max record byte size $\leq$ k $\leq$ 32,767 The DMSANLZ default is 2048 for disk/RAD files and CP-V labeled tape, and is 4096 for ANS labeled tape.
ANS LABEL FILE	Record in the RPCC that all IFID intermediate files to be created shall be allocated to ANS labeled tape, or to CP-V labeled tape, or to disk/RAD files. The DMSANLZ default is ANS labeled tape.
SAVE RELEASE	Record in the RPCC that DMSREST shall save all the intermediate files that it creates. Alternatively, record in the RPCC that DMSREST shall release all intermediate files upon completion of the last process scheduled to use each file. The DMSANLZ default is SAVE. Note that this option does not apply to IFID files SDBA, TDBA, and CTDG, which are always saved.
CHECKSUM NOCHECKSUM	Record in the RPCC that all IFID intermediate files to be created shall be checksummed. Alternatively, record in the RPCC that checksumming shall not be performed. The DMSANLZ default is CHECKSUM.
CIPHER NOCIPHER	Record in the RPCC that all IFID intermediate files to be created shall be enciphered. Alternatively, record in the RPCC that enciphering shall not be performed. Note that this option is meaningless if enciphering was not present in the source database. The DMSANLZ default is CIPHER.
TIME NOTIME	Record in the RPCC that all IFID intermediate files to be created shall include the time as part of the file name. Alternatively, record in the RPCC that time shall not be used. The DMSANLZ default is NOTIME.

Table 11. Diagnostic Messages for Option Errors

Message	Meaning
UNRECOGNIZED OPTION	The scanned characters do not constitute a valid option.
DUPLICATE OPTION	The current option duplicates or contradicts a previously scanned option.
UNRECOGNIZED DIGIT	A character following an equal sign is not numeric.
INVALID OPTION VALUE	The value following an equal sign is not within the allowed range for the option.
SYNTAX COMMA MISSING	Required comma separating two options has been omitted.

7. DATABASE RESTRUCTURING PROCESSOR

The Database Restructuring Processor (DMSREST) restructures a source database into a target database that has been initialized by use of the DMSINIT utility. Restructuring is accomplished by executing those processes that have been scheduled and stored in the RPCC by DMSANLZ. All processes for which nonnull files exist are executed by DMSREST in the order in which they were scheduled by DMSANLZ. This step-by-step execution of scheduled processes is entriely automatic.

The execution of scheduled processes consists of four phases. Phase 1 unloads the source database and prepares the data required to load the target database. Phase 2 loads the target database. Phase 3 prepares data needed to relink all reference codes, and then relinks and validates the target database. Phase 4 reports any errors detected by DMSREST during the first three phases. Appendix I shows where shceduled IFID files are created, the flow of data through all four phases, and all possible processes that may be scheduled by DMSANLZ.

DMSREST Operational Interface

The DMSREST Processor may be executed in a batch mode or on-line from a terminal. The operation of DMSREST relative to the number of scheduled processes to be executed is determined by control command options. The specification of some of these options is recorded in the RPCC. The computer operator or terminal user may also influence the execution of DMSREST via console interrupts and keyins. User-assigned DCBs control the allocation of all intermediate files to public and private storage volumes. Allocation of all intermediate files is catalogued in the RPCC. DMSREST includes operational features that provide for execution breakpoints and restart, along with execution fault protection through backup/recovery procedures.

Each of the DMSREST operational interfaces is described below.

!DMSREST Control Command

The batch monitor control command to execute DMSREST has the form:

```
!DMSREST [option][,...]
```

The corresponding on-line command is:

```
!DMSREST
```

DMSREST prompts the on-line user with a colon (:) for any options. A carriage return or line feed is regarded as indicating the end of the list of options.

Options may be specified in any order, but no option may be repeated. The options and their meanings are listed in Table 12. Note that some options cause information to be stored in the RPCC. Once an option has been recorded in the RPCC, it remains in effect until a subsequent execution of DMSREST respecifies the option. The !DMSREST command and options are listed via the M:LO DCB. Any errors in the command are listed via the M:DO DCB. Table 11 lists the diagnostic messages for option errors.

If the !DMSREST command is given without options, each of the scheduled processes is executed, in the order specified by DMSANLZ, until all have been executed or until execution is interrupted by the terminal user or computer operator.

Ordinarily, execution can be resumed after interruption by a !DMSREST command without options. If DMSREST is interrupted during the re-execution of a step number range of processes, however, the originally specified RERUN range is not completed. To complete the range after an interruption, the user must specify the balance of processes to be re-executed. This can be accomplished via the RERUN and BREAK options in a subsequent job step.

Table 12. DMSREST Options

Option	Meaning
LOG NOLOG	Record in the RPCC that all future processes to be executed shall be logged on the operator's console. Alternatively, record in the RPCC that logging shall not be performed. The DMSANLZ default is LOG.
RUN = n	Breakpoint process execution after n steps, where last process step executed + n ≤ highest process step number. The first process executed is the next scheduled process not yet executed. RUN is an illegal option when RERUN or BREAK options are specified. If n is zero, no scheduled processes are executed, but the exercise of some other options which may access the RPCC is permitted. The DMSREST default is to run all scheduled processes not yet executed.
RERUN = r	Re-execute processes beginning with process step r, where 1 ≤ r ≤ last process step executed. All processes beginning with step r through the highest process step number are executed unless an option or operator BREAK intervenes.
RELOAD = a	Re-execute all processes that load target database area number a, where 1 ≤ a ≤ 64. RELOAD is an illegal option when the RERUN option is also specified. The user must reinitialize the target database area using DMSINIT before specifying this option. DMSREST will re-execute the processes necessary to load the target database area and then continue normal process execution. Concurrent specification of the BREAK or RUN options are only effective during this continued normal process execution.
BREAK = b	Breakpoint process execution after process step b, where last process step executed < b ≤ highest process step number. RERUN in conjunction with BREAK permits a range of process steps to be re-executed. If RERUN is specified, then r ≤ b ≤ highest process step number.
WORK = w	Record in the RPCC that all future set recoveries to be executed shall employ w work file DCBs, where 3 ≤ w ≤ 8. The DMSANLZ default is 8.

Table 12. DMSREST Options (cont.)

Option	Meaning
SORT = s	Record in the RPCC that all future sorts to be executed shall employ s work file DCBs, where $3 \leq s \leq 17$. The DMSANLZ default is 17.
BUFFERS = $\begin{Bmatrix}1\\2\end{Bmatrix}$	Record in the RPCC that either one or two buffers may be allocated to each IFID intermediate file, and the source and target database areas. The DMSANLZ default is 2.
BLOCK = k	Record is the RPCC that all future IFID intermediate files to be created shall have a nominal block size of k bytes, where 8 + max record byte size $\leq k \leq$ 32,767. The DMSANLZ default is 2048 for disk/RAD files and for CP-V labeled tape, and is 4096 for ANS labeled tape.
ANS LABEL FILE	Record in the RPCC that all future IFID intermediate files to be created shall be allocated to ANS labeled tape, or to CP-V labeled tape, or to disk/RAD files. The DMSANLZ default is ANS.
SAVE RELEASE	Record in the RPCC that DMSREST shall save all intermediate files that it creates. Alternatively, record in the RPCC that DMSREST shall release all intermediate files upon completion of the last process scheduled to use each file. The DMSANLZ default is SAVE. Note that this option does not apply to IFID files SDBA, TDBA, or CTDG which are always saved.
CHECKSUM NOCHECKSUM	Record in the RPCC that all future IFID intermediate files to be created shall be checksummed. Alternatively, record in the RPCC that such files shall not be checksummed. The DMSANLZ default is CHECKSUM.
CIPHER NOCIPHER	Record in the RPCC that all future IFID intermediate files to be created and enciphered shall be enciphered. Alternatively, record in the RPCC that such files shall not be enciphered. Note that this option is meaningless if enciphering was not present in the source database. The DMSANLZ default is CIPHER.
CATALOG	List through the M:LL DCB all saved IFID files. Any volume serial numbers for each IFID file are listed. The listing is attached to the L1 logical device for immediate output and is produced before any scheduled process is executed. An example of the listing is shown in Appendix J, Figure J-10.
TIME NOTIME	Record in the RPCC that all IFID intermediate files to be created shall include the time as part of the file name. Alternatively, record in the RPCC that time shall not be used. The DMSANLZ default is NOTIME.

Breakpoint/Restart

Breakpoint/restart features permit a database to be restructured in a series of monitor job steps by selective execution of scheduled processes. A breakpoint occurs in DMSREST at the completion of each scheduled process and before execution of the next scheduled process. DMSREST completes each scheduled process with the following activities:

1. All IFID files created are closed and catalogued in the RPCC.
2. The executed process is marked "completed" in the RPCC.
3. The RPCC is closed.
4. A checkpoint RPCC file is created to enable DMSREST recovery.

If a breakpoint is not active at the end of the process, DMSREST will then open the RPCC and execute the next scheduled process.

Activation of a breakpoint is controlled by the control command options RUN or BREAK, as described earlier. Either option may be used to select the number of scheduled processes DMSREST shall execute during a job step. The computer operator may also activate the next breakpoint by use of the operator keyins described below. An operator activated breakpoint has priority over any activated by a control command option. DMSREST terminates when an activated breakpoint occurs.

Restart from an activated breakpoint is accomplished by re-executing DMSREST in a batch or on-line mode. DMSREST examines the RPCC to determine the last process completed prior to the activated breakpoint. It will then automatically begin execution with the next scheduled process. New breakpoints may be activated when DMSREST is restarted.

Backup/Recovery

DMSREST includes functions that permit recovery and restart of an abnormally terminated job step. These are the RPCC checkpoint file and the RERUN and RELOAD control command options.

The RPCC checkpoint file is created by DMSREST after the successful execution of each process. The file is named DMSRESTRPCCHKPT. Creation of a new checkpoint file automatically releases the checkpoint file created by the previous process.

Recovery and re-execution of a partially completed job step is performed by restarting DMSREST. When DMSREST is restarted it will examine the current contents of the RPCC. If the RPCC was not closed properly after the last executed process, DMSREST will automatically recover the RPCC using the current RPCC checkpoint file. Execution will then proceed with the process that was in execution when the job step terminated. DMSREST will output the following message:

```
DMSREST AUTOMATICALLY RECOVERED FROM INCOMPLETE PROCESS
EXECUTION IN STEP n
```

Recovery of the restructuring process may be necessary if an intermediate file created by DMSREST cannot be accessed. The control command option RERUN may be used to re-execute previously completed processes and thus recreate the intermediate files. When a process is re-executed, all scheduled output IFID files are recreated and catalogued in the RPCC. Cataloging of the files in the RPCC replaces those catalogued during the previous execution of the process. The previous output IFID files, however, are not released by DMSREST if the TIME option was in effect. Re-execution of scheduled processes continues until an activated breakpoint occurs, at which time DMSREST will terminate.

DMSREST executes two types of processes that place information in the target database. These are the LOAD and LINK processes. One or more LOAD processes and one LINK process will be scheduled for each target database area. The execution of each of these processes depends upon information placed in the target database by the

previous process; it is impossible, therefore, to restart DMSREST if a LOAD or LINK process was partially completed. If this type of restart is attempted, DMSREST will output the following message and then terminate.

```
DMSREST REQUIRES INITIALIZATION OF TARGET AREA n name
RE-EXECUTE DMSREST WITH RELOAD = n CONTROL COMMAND
```

The user must first reinitialize target area n using DMSINIT. DMSREST may then be restarted with the RELOAD option. This will cause DMSREST to re-execute all LOAD steps already completed for the specified area. After these steps are re-executed, processing will then continue beginning with the process that was in execution when the job step terminated.

Operator Communication

DMSREST accepts operator interrupts and keyins to facilitate run-time control when executing in batch mode or from an on-line terminal. If DMSREST accepts the control command options, it will then output the following message on the OC device in batch mode, or on the user's terminal.

```
DMSREST - WILL ACCEPT OPERATOR INTERRUPTS
```

This message is not produced if the NOLOG option is active.

The operator may then interrupt DMSREST to establish communications. In batch mode, the INT[†] keyin is used to effect an interrupt. Depressing the BREAK key causes an interrupt when DMSREST is executing from an on-line terminal. Upon receipt of an interrupt, DMSREST responds with the following message:

```
DMSREST - INTERRUPTED BY OPERATOR, YOU MAY INPUT:
```

Operator keyins are then solicited by the following multi-line message:

```
IGNORE   - TO IGNORE YOUR INTERRUPT
PAUSE    - TO PAUSE DMSREST AFTER CURRENT STEP
BREAK    - TO TERMINATE DMSREST AFTER CURRENT STEP
GO       - TO CANCEL YOUR PRIOR 'PAUSE' OR 'BREAK'
CATALOG  - TO LIST ALL FILES AFTER CURRENT STEP
QUIT     - TO IMMEDIATELY TERMINATE DMSREST
```

DMSREST then requests a keyin by prompting with the message:

```
DMSREST - YOUR INPUT:
```

DMSREST responds to invalid keyins with the following message:

```
DMSREST - YOUR RESPONSE IS INVALID, YOU MAY INPUT:
```

The operator may then type a corrected keyin.

Keyins may be preceded or followed by blank characters; only the first 15 characters, however, are accepted by DMSREST. Operator keyins that may be entered, and the resulting actions and messages by DMSREST, are given in Table 13.

[†]See the CP-V Operations Reference Manual (90 16 75).

Table 13. DMSREST Keyins and Responses

Operator Keyin	DMSREST Response
IGNORE	DMSREST returns to the interrupted process without taking any action other than issuing the message: `DMSREST - YOU'RE IGNORED`

PAUSE	DMSREST activates a pseudo breakpoint after the currently executing process. It then issues the message:
	`DMSREST - WILL PAUSE AFTER STEP n`
	When the pseudo breakpoint is reached, DMSREST will issue the following message at five minute intervals:
	`DMSREST - PAUSE FOR OPERATOR INTERRUPT AFTER STEP n`
	The operator may then interrupt DMSREST and enter another keyin. The PAUSE keyin cancels any previous BREAK keyins.

BREAK	DMSREST activates a breakpoint after the currently executing process and then issues the message:
	`DMSREST - BREAKPOINT SET AFTER STEP n`
	When the breakpoint is reached, DMSREST issues the following message and then terminates execution.
	`DMSREST - OPERATOR BREAKPOINT TERMINATION BEFORE STEP n`
	The BREAK keyin cancels any previous PAUSE keyin.

GO	DMSREST deactivates any breakpoint activated by a previous PAUSE or BREAK keyin and then issues the following message:
	`DMSREST - YOUR PAUSE OR BREAKPOINT CANCELLED`
	If no prior PAUSE or BREAK keyin was entered, DMSREST issues the message:
	`DMSREST - NO PRIOR PAUSE OR BREAKPOINT SET - YOU MAY INPUT`
	The operator should then enter some other keyin. Breakpoints activated by the control command options RUN or BREAK are not cancelled by a GO keyin.

Table 13. DMSREST Keyins and Responses (cont.)

Operator Keyin	DMSREST Response
CATALOG	DMSREST prepares to produce a file catalog listing and then issues the following message: DMSREST - FILES WILL BE LISTED AFTER STEP n When step n is completed, DMSREST will produce a file catalog listing and then continue execution with the next scheduled process. The catalog listing is attached to the L1 logical device for immediate output. A sample of the catalog listing is provided in Appendix J.
QUIT	DMSREST issues the following message: DMSREST - TERMINATED BY OPERATOR The DMSREST processor is then immediately aborted.

DCB Assignments

No interface is required between the DMSREST user and the CP-V monitor, with respect to the assignment of the RPCC, IFID files, sort work files, or listing output DCBs. Table 14 lists all DMSREST output DCBs and the contents of the associated files. IFID input DCBs are dynamically assigned by DMSANLZ.

If the RPCC is assigned to a private disk pack when it is created by DMSANLZ, an !ASSIGN command (or !SET command, on-line) is required for the F:RPCC DCB. The private pack serial number is entered with the SN parameter of the !ASSIGN command. The file name must be DMSRESTRPCC. An example is shown below.

```
!ASSIGN F:RPCC, (FILE, DMSRESTRPCC), (SN, DISK)
```

DMSREST defaults all output IFID files to ANS labeled tape. This default assignment may be changed by use of the LABEL or FILE option in the !DMSREST control command (see Table 12). The user may temporarily override the default assignment during a given job step by use of an !ASSIGN command for an output IFID DCB. The output IFID DCBs are F:WF01 through F:WF15. Only the !ASSIGN command keywords FILE, LABEL, and ANSLBL are permitted for these DCBs. DMSREST will issue an error diagnostic and abort if any !ASSIGN command for an IFID DCB specifies the DEVICE keyword. Note that standard DMSREST file names replace the file names specified in !ASSIGN commands for IFID DCBs. Only one IFID file is written to any tape volume, but many IFID files may be output to a private disk volume set. An example of output IFID file assignment for a job step is shown below. In this example, all files output through F:WF01 during the next job step will be public disk/RAD files, those output through F:WF02 will be on CP-V labeled tape, and those output through F:WF03 will be on ANS labeled tape.

```
!ASSIGN F:WF01, (FILE, WF01)

!ASSIGN F:WF02, (LABEL, WF02)

!ASSIGN F:WF03, (ANSLBL, WF03)
```

The Xerox Sort program, used by DMSREST, defaults all sort work files to public files. See the Xerox Sort and Merge (for CP-V/BPM) Reference Manual (90 11 99) for information on assigning sort work file DCBs.

Table 14. DMSREST DCBs and File Contents

DCB	Mode	Contents
F:RPCC	inout	RPCC
F:WF01	out	CIDG, CSMR
F:WF02	out	CCDG
F:WF03	out	CVDG
F:WF04	out	CSKD, CSTO, CGOR
F:WF05	out	CRCO
F:WF06	out	CEGR, CGRC
F:WF07	out	CTDG
F:WF08	outin	WRK1
F:WF09	outin	WRK2
F:WF10	outin	WRK3
F:WF11	outin	WRK4
F:WF12	outin	WRK5
F:WF13	outin	WRK6
F:WF14	outin	WRK7
F:WF15	outin	WRK8
F:SCRF1	outin	Xerox Sort intermediate file
.	.	.
.	.	.
.	.	.
F:SCRF17	outin	Xerox Sort intermediate file
M:LO	out	Listing output
M:DO	out	Diagnostic output
M:LL	out	Listing log (for control command and operator keyins requesting file catalog listings)

Volume serial numbers used by IFID files output on ANS labeled tape, CP-V labeled tape, and private disk packs are copied from the IFID output DCB into the RPCC when each IFID file is closed. Volume serial numbers are entered into DCBs by an !ASSIGN command or by MOUNT or ANSMOUNT keyins in response to a monitor message (see the CP-V Operations Reference Manual, 90 16 75). DCBs F:WF01 through F:WF15 each permit up to 10 volume serial numbers.

The SN parameter of an !ASSIGN command for DCBs F:WF01 through F:WF15 may preallocate up to 10 volume serial numbers. Any IFID file output through an assigned DCB may use some number of these preallocated volumes. Tape volumes may contain only a single IFID file, but private disk packs may contain many IFID files.

Tape volume serial numbers for each assigned DCB are saved by DMSREST. When an output IFID uses a DCB, DMSREST reallocates to that DCB any saved, unused tape volume serial numbers that were preallocated to the DCB by an !ASSIGN command. After the IFID file is closed, all unused tape volume serial numbers are again saved by DMSREST and become available for allocation to some other IFID file that may be output through that DCB. An example is shown below.

```
!ASSIGN F:WF04, (ANSLBL, WF04), (SN, 111111, 222222, 333333, 444444)
```

In this example, if a CSKD output IFID file created in step 1 uses tape•111111, then volume serial numbers 222222, 333333, and 444444 are available for the next IFID file output through F:WF04. If another CSKD file output in step 2 uses tapes 222222 and 333333, then only the preallocated volume serial number 444444 is available for the next IFID file output through F:WF04. If a CSKD file output in step 3 needs a second tape, then 444444 is the first volume and whatever tape is mounted by the operator is the second volume. Any IFID files output through F:WF04 in subsequent steps will also have serial numbers of tapes mounted by the operator.

Private disk file volume serial numbers may accommodate multiple IFID files. The monitor allocates each file to those volumes of a private volume set that have unused space. DMSREST only records in the RPCC the first volume serial number of a private volume set containing an output IFID file. A private volume set may also be extended by means of an !ASSIGN command, as shown below.

```
!ASSIGN F:WF03, (FILE, WF03), (SN, 1111, 2222, 3333)
```

In this example, if a CVDG IFID file output in step 1 is wholly contained by disk volume 1111, then this volume serial number is recorded in the RPCC. If another CVDG file output by step 2 is spread across disk volumes 2222 and 3333, then the first volume serial number of the private volume set is recorded in the RPCC, namely 1111. Note that CP-V uses the first volume serial number of a private volume set to locate a file on any volume or volumes of that set.

Another example of private volume set allocation is shown below.

```
!ASSIGN F:WF03, (FILE, WF03), (SN, 1111, 4444, 5555)
```

In this example, if the private volume set was 1111, 2222, 3333, then the !ASSIGN command extends the set to include volumes 4444 and 5555. An output IFID file may be allocated by the monitor anywhere within this five-pack set of volumes. DMSREST only records the first volume (1111 in this example) in the RPCC for any IFID file output to the extended private volume set.

DCBs M:LO and M:DO use vertical format control and may be assigned to logical devices, files, or tape. The M:LO DCB lists all processes executed and files created; the M:DO DCB lists all error diagnostics. An example of M:LO assignment is shown below.

```
!ASSIGN M:LO, (FILE, SAVELO)
```

In this example, all listings output through M:LO in the next job step will go to a public file named SAVELO.

File catalog listings are output through M:LL in response to control command options and operator keyins. DMSREST connects M:LL to the L1 logical device, to enable immediate release of the symbiont file to an output device. Vertical format control is present in the output.

8. APL/EDMS INTERFACE

A special interface allows APL programs access to EDMS databases. The interface consists of an intrinsic function, within APL, and a set of APL functions that perform argument formatting operations and execute the intrinsic function. These functions normally reside in workspace DMSFNS in the DMSLIB account. See Figure 13 for a graphic representation of the APL/EDMS interface.

Since all user-function communication with the EDMS intrinsic function will be accomplished via the functions in the DMSFNS workspace, the material in this chapter describes these functions only, and not the intrinsic function itself. It is assumed that the reader is familiar with both APL and the services provided by the EDMS Database Manager (DBM), as described in Chapter 4.

The functions described in this chapter provide the APL programmer with most of the EDMS services available to the COBOL, FORTRAN or assembly language programmer. Not available to the APL programmer are

FINDSEQ ENDSTATS

DMSRTRN RPTSTATS

DMSSTATS

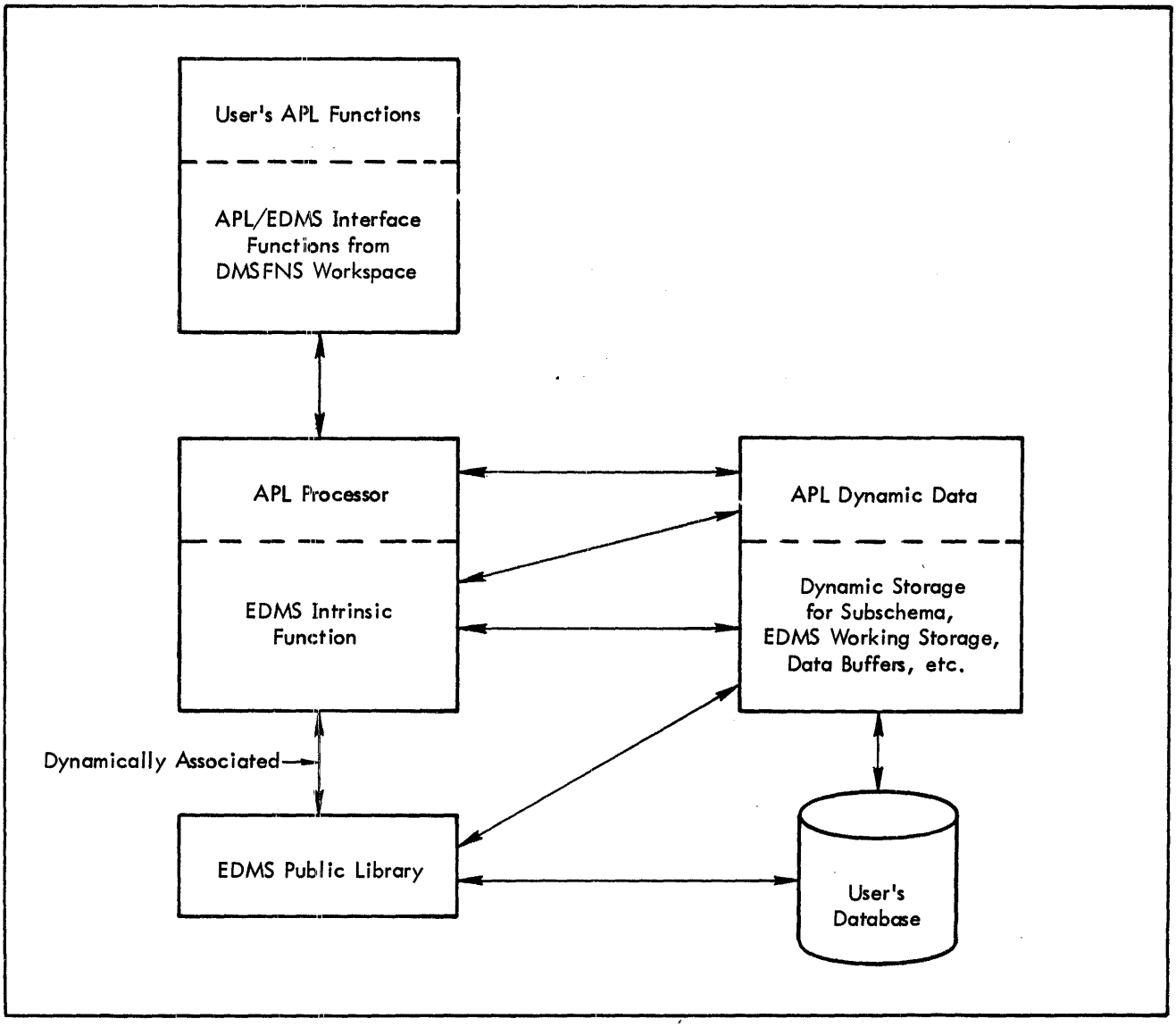


Figure 13. APL/EDMS Interface

Additionally, functions that provide services unique to the APL environment include:

DMSSUB	TODMS
DMSPKSN	FROMDMS
DMSPASS	CURRGRP
REFCODE	CURRSET
FRSTREF	DMSEND
LASTREF	DMSERCOD
BREFCODE	ECDREF
BGRPNO	DCDREF
BERRCODE	

APL/EDMS Overview

The first time an EDMS function is called, the EDMS intrinsic function within APL associates the EDMS Public Library and allocates dynamic memory to contain the user's subschema, working storage, name table and data buffers. This dynamic storage is released, and the EDMS Public Library is disassociated, when the user executes a DMSEND call or successfully executes an APL Load Clear, Off, Save or Continue command.

Unless otherwise stated, the functions described in this chapter will return a null integer vector instead of a meaningful, explicit result. Typically, however, there would still be an implicit result of changes to the database and to the user's logical "position" within the database structure, which would be reflected in the contents of the EDMS working storage maintained by the Database Manager (DBM). These values are available to the user-functions through the use of special working storage communication functions.

Errors

Two levels of errors are possible when using the APL/EDMS interface: EDMS-level and APL-level errors. EDMS-level errors are the standard EDMS errors, as defined in Appendix F. These are handled as they would be for FORTRAN, COBOL and assembly language programs; i.e., the ERR-CODE cell in the Communication Control Block (CCB) is set to indicate the type of error encountered. APL-level errors are errors detected by the EDMS intrinsic function, and are handled in a manner similar to APL domain, rank and length errors.

Character Translations

Because of the variety of terminal types that can be used for APL all input characters are translated into an internal APL code. The only character translation of importance to the APL/EDMS interface is that of the hyphen as used in EDMS data names. As an aid to the APL/EDMS user, all functions in the DMSFNS workspace will automatically translate hyphens in data-name text vectors to the appropriate code for use by the EDMS intrinsic. No translation, however, is done on character data values. It is the user's responsibility to insure that character data going into EDMS working storage (i.e., via the TODMS function) have the correct hexadecimal configuration. Appendix B of the APL reference manual describes the internal APL character set.

Item Identifiers

The item identifier is a special form of argument accepted by the TODMS, FROMDMS and FINDX functions. It consists of the item name, optionally qualified by a group name or set name. Examples:

'QTY-ON-HAND'	No qualification
'DATE IN STATUS-GRP'	Group qualification
'PARTNO OF WHERE-USED-SET'	Set qualification

The prepositions 'of' and 'in' are interchangeable in all qualification formats.

Group-name qualification is required only when an item name is not unique. Set-name qualification is a mechanism that allows the user to reference alias working storage areas. In this case, the named set must be one on which the subject item has an alias defined. Note that the named item resides in the group defined as owner of the named set. Item identifiers are always used in the form of text vectors.

Reference Codes

Two reference code formats are recognized by the functions in the DMSFNS workspace: encoded and decoded. A decoded reference code is a three-element integer vector consisting of area number, page number, and line number. An encoded reference code is an integer scalar that is the hexadecimal equivalent of a standard 32-bit reference code, with the area number in bits 0 through 7 but limited to 64_{10}, and page and line numbers sharing the remaining 24 bits. Thus, the integer value of an encoded reference code is in the range 2^{24} through 64×2^{24}.

The only function that recognizes reference codes in decoded format is ECDREF, which converts decoded reference codes to encoded format. All other functions that accept reference code arguments require them in encoded format.

Function Usage

Functions accomplish all user-function interaction with the database and EDMS working storage. The function descriptions given in the following paragraphs assume that the user is familar with both the APL language syntax and EDMS services described in Chapter 4.

Beginning of Processing

Before manipulating any data, the user must identify the subschema for the database he intends to process, open the database, and specify volume serial numbers and EDMS password.

Identifying the Subschema

DMSSUB, used to identify the subschema, is a monadic function whose argument is a text vector composed of subschema name and, optionally, account and monitor password, separated by periods.

Format

```
DMSSUB 'subschema name [.[account] [.password]]'
```

Examples

DMSSUB 'TESTSUB'	Subschema name only
DMSSUB 'PERSONNEL..PRSPASS'	Subschema name and monitor password only
DMSSUB 'PARTSUB.MFGACCT'	Subschema name and account only
DMSSUB 'HOSPSUB.HOSPITAL.PASS01'	Name, account, and password

Usage Rules

1. DMSSUB must be the first EDMS function that is executed, except for the error control functions which may be executed at any time.
2. The specified subschema must include a name table.
3. The argument text vector must not contain any blanks.
4. The account and monitor-password portions of the argument are optional.
5. The DMSSUB call is legal at any time when there are no open database areas.

Response

DBM places subschema file identification information in the DCB through which the subschema will be read, and associates the public library. On the first EDMS call after the DMSSUB call (with the exception of the DMSPKSN call, described below), the EDMS intrinsic function acquires dynamic memory to contain the subschema, working storage, name table and three data buffers.

Openning the Database

The functions used by APL to open the database are the same as the open-calls described in Chapter 4. Function type is monadic. Argument is a text vector containing area name and, optionally, account, monitor password, and cipher key, separated by periods.

Format

{OPENUPD / OPENRET / OPUPDSHD / OPRETSHD / CREATE} 'area name[.[account][.[monitor password][.cipher key]]]'

Examples

OPENRET 'TESTAREA'	no account, monitor password or cipher key
OPENUPD 'PAYROLL..PAYPASS.XPAY'	monitor password and cipher key
OPUPDSHD 'PARTSDB.MFGACCT'	account only
OPRETSHD 'HOSPDB...HOSP'	cipher key only

Usage Rules

1. Database open functions are subject to the same usage rules as their standard EDMS counterparts.
2. Indicated area must be an existing EDMS database area file that is defined in the subschema.
3. Account, monitor password and cipher key portions of argument are optional.
4. Database open functions are legal any time after the DMSSUB call as long as there are no active database areas.

Response

In addition to the standard open procedures described in Chapter 4, the EDMS intrinsic function acquires dynamic memory for an inventory buffer, if required, and allocates an APL file I/O DCB for use in accessing the area.

Specifying Volume Serial Numbers

DMSPKSN is a monadic function, whose argument is a text vector containing volume serial numbers.

Format

DMPKSN 'volume-serial-numbers'

Usage Rules

1. The argument must be text vector, from 1 to 12 characters in length containing volume serial numbers.

2. If a pack serial number assigned to, or contained in, the argument text vector consists of fewer than four characters and is not the last (or only) serial number in the string, then the user must pad that serial number with trailing blanks to make its length an even four characters.

3. No intervening EDMS calls are allowed between a DMSSUB or open call and its associated DMSPKSN call.

Response

If the length of an argument test vector is not an even multiple of four, the EDMS intrinsic function pads it with trailing blanks to make it so. The first four characters of the padded argument are then taken as the first volume serial number, the second four as the second volume serial number, and the last four as the third. The EDMS intrinsic function then applies the serial numbers to the most recent DMSSUB or open call

Example

In the following example the subschema and two database areas reside on the private volume set consisting of PK1, PK2 and PK3. Note the trailing blank after the first two volume serial numbers.

```
DMSSUB 'TESTSUB'

DMSPKSN VSNS←'PK1 PK2 PK3'

OPENRET 'TESTAREA1'

DMSPKSN VSNS

OPENUPD 'TESTAREA2'

DMSPKSN VSNS
```

Specifying an EDMS Password

DMSPASS is a monadic function whose argument is a text vector containing the EDMS password.

Format

DMSPASS 'password'

Usage Rules

1. The argument must be a text vector consisting of a string zero to eight characters long. A null vector causes the password portion of the Communication Control Block (CCB) to be set to blanks.

2. The DMSPASS function may be used at any time after the DMSSUB call and before the first access to the database.

Response

The EDMS intrinsic pads the argument with trailing blanks to a length of eight characters, as required, and moves it into the PASSWORD cell of the CCB.

Updating the Database

Updating Group Occurrences

APL functions for adding and deleting group occurrences in the database are subject to the same usage rules as their standard EDMS counterparts. A difference exists in modifying data values. From APL this can be done only for an entire group, whereas other host-language procedures can modify selected items within a group. The APL updating functions are monadic. Argument is a text vector containing the group name.

Format

```
{ STORE     }
{ DELETE    }
{ DELETAUT  }
{ DELETSEL  } 'group-name'
{ REMOVE    }
{ REMOVSEL  }
{ MODIFY    }
```

Modifying Set Linkages

The occurrence of a group whose membership in a set is defined as optional or manual can be linked to, or delinked from, a set occurrence. Also, a member group occurrence can be changed from one owner occurrence to another in any set in which it participates. The functions to accomplish these modifications are dyadic and subject to the same usage rules as the equivalent standard EDMS calls. Both arguments are text vectors.

Format

```
'group-name' { LINK   } 'set-name'
             { DELINK }
             { RELINK }
```

Group-Relative Retrieval

The group-relative retrieval functions are equivalent to their standard EDMS counterparts and subject to the same usage rules. They are monadic functions. Argument is a text vector containing group-name.

Format

```
{ FINDC    }
{ FINDG    }
{ FINDDUP  }
{ FINDFRST } 'group-name'
{ FINDLAST }
{ FINDN    }
{ FINDP    }
```

Response

The specified occurrence of the named group is retrieved according to the criteria implicit in the type of call. When the last applicable group occurrence has been retrieved as the result of a FINDN or FINDP call, DBM sets the GRP-NO cell in the CCB to zero as an indication of the "at-end" condition.

Usage Rules

1. For the FINDFRST, FINDLAST, FINDN, and FINDP calls, the indicated group must be defined with INDEXED location mode. Note that the FINDN and FINDP calls also accept set-name arguments when used to traverse sets (see below).

2. The group-relative retrieval functions are subject to the same usage rules as their standard EDMS counterparts.

Set Traversal

The functions are equivalent to their standard EDMS counterparts and subject to the same usage rules. They are monadic. Argument is a text vector containing set-name.

Format

```
{FINDN}
{FINDP}
{FINDM} 'set-name'
{HEAD }
```

Response

DBM traverses the named set in the direction implicit in the type of call. Note that the HEAD function includes an implicit move of the data items to working storage.

Retrieving Secondary-Index Items

FINDX, used for retrieving secondary indexes, is subject to the same usage rules as the standard EDMS FINDX procedure. It is a monadic function whose argument is a text vector containing an item identifier. The format of the item identifier was given earlier in this chapter (under "Overview").

Format

```
FINDX 'item-name[{OF} group-name]'
                {IN}
```

Response

DBM retrieves the first (or next) secondary index for the indicated item. When the last applicable secondary index occurrence has been retrieved, DBM sets the GRP-NO cell in the CCB to zero as an indication of the "at-end" condition.

Direct Retrieval

The direct retrieval functions are nyladic and subject to the same usage rules as their standard EDMS counterparts.

Format

```
{FINDD }
{FINDS }
{FINDSI}
```

Response

DBM retrieves a group occurrence on the basis of a reference code in the CCB according to the method implicit in the type of call. When the last applicable group occurrence has been retrieved as the result of a FINDS or FINDSI call, DBM sets the GRP-NO cell in CCB to zero as an indication of the "at-end" condition.

Moving to Working Storage

The GET function is monadic and subject to the same usage rules as the standard EDMS GET procedure. The GET function's argument is a text vector containing group name.

Format

```
GET 'group-name'
```

Response

DBM moves the contents of all items in the most recently retrieved occurrence of the indicated group from the DBM data buffers to EDMS working storage. There is no facility for moving selected items.

Communicating with Working Storage

The working storage communication functions let the user (1) initialize selected areas of the EDMS CCB and working storage, and (2) obtain specific values from the CCB and working storage areas in the form of an APL result.

CCB Initialization

The CCB initialization functions are monadic. Argument is an integer scalar reference code, in encoded format, or a retrieval count. (Encoding and decoding of reference codes is explained below under "Reference Code Conversions".) Reference code arguments will typically be the result of either the ECDREF or BREFCODE functions.

Format

```
{FRSTREF  }
{LASTREF  }
{REFCODE  }
```

Examples

1. Saving and restoring the reference code of a specific group occurrence.

```
SAVEREF←BREFCODE
   :
   :
REFCODE SAVEREF
```

2. Retrieve all group occurrences on page 1 of area 3. (First column is area; 2nd, page number; 3rd, line number.)

```
FRSTREF  ECDREF  3  1  1
LASTREF  ECDREF  3  1  255
```

3. Retrieve ten group occurrences starting at page 12 of area 1.

```
FRSTREF  ECDREF  1  12  1
LASTREF  10
```

Response

The EDMS intrinsic function places the value of the argument in the indicated cell of the CCB.

Usage Rules

1. The retrieval count form of the argument is acceptable only by the LASTREF function. It is intended for use in conjunction with the FINDSI function.
2. The CCB initialization functions may be used at any time after the DMSSUB call and before a DMSEND call.

CCB Inquiry

The purpose of the CCB inquiry functions is to obtain an integer scalar representing the contents of the REF-CODE, GRP-NO, or ERR-CODE cell of the CCB, as indicated by the function name specified.

Format

```
{BREFCODE  }
{BGRPNO    }
{BERRCODE  }
```

The functions are nyladic. They can be used at any time after the DMSSUB call and before a DMSEND call.

Data Item Initialization

TODMS is a dyadic function whose left argument is a data value and whose right argument is a text vector containing an item identifier. The data value must be either an APL variable or the result of an APL expression. Item name, optionally qualified by group-name or set-name, constitutes the item identifier. The data value is moved to the EDMS working storage area for the indicated item.

Format

value TODMS 'item-name$\left[\left\{\begin{matrix}\underline{\text{OF}}\\\underline{\text{IN}}\end{matrix}\right\}\left\{\begin{matrix}\text{group-name}\\\text{set-name}\end{matrix}\right\}\right]$'

Usage Rules

1. If the item type is defined so as to be interpreted by EDMS as alphabetic or alphanumeric, then the data type of the left argument must be character or a domain error will result (see Table 1, Chapter 3). If the item definition were to result in an EDMS interpretation other than alphabetic or alphanumeric, the left argument must be real, logical or integer. The TODMS function automatically converts the data value to the type defined by the DDL. Truncation errors are possible whenever a real APL value is moved to a binary or floating short EDMS item, or whenever the object of the TODMS call is a numeric or packed decimal item. Truncation is defined as any loss of integral or significant functional digits.

   Conversions to numeric or packed decimal items are rounded to the number of digits specified by the item's picture with a maximum of 15 digits. All conversions involving floating point numbers yield results identical to those of APL; however, this does not guarantee that converted numbers will exactly match those produced via another processor.

2. The shape and dimensions of the data value must conform to the DDL description of the EDMS item. In no event is an array of greater than two dimensions acceptable.

   For alphabetic and alphanumeric EDMS items that do not contain an OCCURS clause, the data value must be either a scalar or a vector. A scalar is acceptable only for single-character items (e.g., PIC X). For items containing more than a single character, a character-vector data value is required. The length of the vector must be exactly the same as that of the EDMS item. For example, an item defined as PIC X(12) requires a twelve element character vector as left argument for TODMS. If the EDMS item description contains an OCCURS clause, the data value must be a matrix with its first dimension equal to the item's occurs count and its second dimension equal to the number of characters in the item description. Example: An item defined as PIC X OCCURS 5 requires a 5 by 1 matrix as the left argument for TODMS.

   For EDMS items whose type is not alphabetic or alphanumeric and that do not contain an OCCURS clause, only a scalar left argument is acceptable. If the EDMS item description contains an OCCURS clause, then the left argument for TODMS must be a vector whose length is equal to the item's occurs count.

3. For numeric or packed decimal EDMS items whose picture does not specify a sign, only a positive value may appear as left argument. Any attempt to move negative values to an unsigned item will result in an APL-level EDMS error.

Table 15 gives examples of illegal left arguments along with a form of the argument that would be legal, or an expression describing the legal range of values that the argument may assume.

Data Item Inquiry

FROMDMS obtains the contents of a data-item from working storage. The form of the result is determined by the item's DDL description with respect to type, rank, and dimensions, in accordance with Tables 16, 17, and 18 shown below. The function is monadic. Its argument is a text vector containing item identifier.

Format

FROMDMS 'item-name$\left[\left\{\begin{matrix}\underline{\text{OF}}\\\underline{\text{IN}}\end{matrix}\right\}\left\{\begin{matrix}\text{group-name}\\\text{set-name}\end{matrix}\right\}\right]$'

Table 15. Examples of Illegal Left Arguments

Item Description	Illegal Left Argument	Reason for Error	Legal Form or Range
PIC X(9)	'ABCD'	INCORRECT LENGTH	9↑'ABCD'
PIC X(8)	2 4⍴'BETA'	INCORRECT RANK	,2 4⍴'ALPHABETA'
PIC X(9) OCCURS 2	'ALPHABETA'	INCORRECT RANK	2 9⍴'ALPHABETA'
PIC XXX	123	INCORRECT TYPE	'123'
PIC 999	'123'	INCORRECT TYPE	123
PIC 99V99	'34.5'	INCORRECT TYPE	34.5
PIC 99V99 OCCURS 3	34.5	INCORRECT LENGTH	3⍴34.5
PIC 99V99	123.4	TRUNCATION	100 > arg
PIC PP999	.01234	TRUNCATION	.01 > arg
BINARY	10*10	TRUNCATION	$-2^{31} \leq arg \leq 2^{31}-1$
FLOATING SHORT	⌈/⍳0	TRUNCATION	7.237E75 ≥ | arg
PIC 999.9	-12	SIGN	12

Table 16. FROMDMS Result: Item Type

EDMS Interpretation† \ APL Type		Character	Real	Integer
Alphabetic or Alphanumeric		X		
Binary				X
Floating Point (Short or Long)			X	
Numeric or Packed Decimal	Fewer than 10 Digits and no 'V' in Picture			X
	10 or more digits or 'V' in Picture		X	

†See Table 1, Picture-Type Correspondences.

Table 17. FROMDMS Result: Item Rank

EDMS Item Description \ Result Rank		Scalar	Vector	Matrix
OCCURS Clause not Specified	Alphabetic or Alphanumeric		X	
	Not alphabetic or Alphanumeric	X		
OCCURS Clause Specified	Alphabetic or Alphanumeric			X
	Not Alphabetic or Alphanumeric		X	

Table 18. FROMDMS Result: Item Dimensions

Item Type / Result Rank	Alphabetic or Alphanumeric	Not Alphabetic or Alphanumeric
Vector	Length is number of characters in item description.	Length is OCCURS Count
Matrix	First dimension is OCCURS count; Second dimension is number of characters in item description.	———

Usage Rules

1. FROMDMS may be used at any time after the DMSSUB call and before a DMSEND call.

2. If FROMDMS encounters a negative number in an unsigned numeric or packed decimal item, the absolute value of the item is returned.

3. If the item's picture specifies a scaling factor that does not allow the item value to be accurately represented in a floating point long number, e.g., PIC P(78)999 or PIC 999(78), an APL-level EDMS error is reported.

4. If an illegal digit is encountered in a numeric or packed decimal item value, an APL-level EDMS error is reported.

5. For numeric or packed decimal items, the maximum significance of the result value is approximately 15 digits. Thus, for items whose picture exceeds 15 digits of significance, e.g., PIC 9(17), only the first 15 digits are represented in the results; however, the scaling is retained.

Table 19 gives examples of the results of FROMDMS.

Table 19. FROMDMS Sample Results

Item's DDL (Description)	Item Contents	FROMDMS Result
PIC 999P(6)	123	1.23E8
PIC P(6)999	123	1.23E-7
PIC 99 OCCURS 5	1122334455	11 22 33 44 55
PIC 9V9 OCCURS 5	1122334455	1.1 2.2 3.3 4.4 5.5
PIC 9(5) OCCURS 2	1122334455	11223 34455
PIC X(10)	ABCDEFG123	A B C D E F G 1 2 3
PIC XX OCCURS 5	ABCDEFG123	AB CD EF G1 23
PIC A(3)X(4)9(3)	ABCDEFG123	A B C D E F G 1 2 3

Current of Type Inquiry

CURRGRP returns the contents of the current-of-type cell for the indicated group as an integer scalar reference code in encoded format. The function is monadic. Its argument is a text vector containing the group-name.

Format

```
CURRGRP 'group-name'
```

Usage Rule

The CURRGRP function may be used at any time after the DMSSUB call and before a DMSEND call.

Examples

```
          CURRGRP 'PART-GRP'
33555095

          DCDREF CURRGRP 'PART-GRP'
     2    5      23
```

DCDREF is the function used for decoding reference codes, explained below under "Reference Code Conversions".

Current of Set Inquiry

CURRSET returns the contents of the set-table for the indicated set as an integer vector of reference codes in encoded format. The function is monadic. Its argument is a text vector containing set-name. CURRSET may be used at any time after the DMSSUB call and before a DMSEND call.

Format

```
CURRSET 'set-name'
```

Example

```
          CURRSET 'CALL-OUT-SET'
16777473  0    16778243    16779321 .

          DCDREF    CURRSET 'CALL-OUT-SET'
     1    1     1
     0    0     0
     1    4     3
     1    8     57
```

DCDREF is the function used for decoding reference codes, explained below under "Reference Code Conversions".

Terminating Processing

The termination functions provide facilities for closing the user's database and releasing dynamic memory.

Closing a Database Area

CLOSAREA closes the area indicated by its argument – a text vector containing the area name. The function is monadic. It is subject to the same usage rules as the standard EDMS CLOSAREA procedure.

Format

CLOSAREA 'area-name'

Closing all Database Areas

CLOSEDB, a nyladic function, causes all database areas to be closed. The function is subject to the same usage rules as the standard EDMS CLOSEDB procedures.

Format

CLOSEDB

Releasing Dynamic Memory

DMSEND, a nyladic function, causes all dynamic memory to be released and the EDMS public library to be disassociated. The function may be used at any time after the DMSSUB call, provided no database area files are open.

Format

DMSEND

EDMS-Level Error-Control Functions

The error-control functions are used to simulate the actions of their standard EDMS counterparts. See "Dynamics of EDMS-Level Error Control", below, for a discussion of the simulation technique.

Setting Error Control for Data-Dependent Errors

SETERR, a dyadic function, specifies the name of the function that is to be invoked if a data-dependent error occurs. The function's left argument specifies the error numbers to be set. The right argument is a text vector containing the name of the function to be invoked if one of the specified errors occurs.

Format

error code(s) SETERR 'function-name'

Examples

(⍳0) SETERR 'ERRORFUNC' Catch and report all data-dependent errors.

3 5 7 SETERR 'CODES357' Catch and report errors 3, 5, and 7.

Usage Rules

1. The error code specification may be in the form of any APL expression that, evaluated, equals an integer scalar or vector with values between 1 and 29 inclusive.

2. The named function must be a nyladic, no-result function. If, at the time a specified error occurs, the named function is defined as other than a nyladic, no-result function, an APL-level error is generated. (See "APL-Level EDMS Errors" below.)

3. The SETERR function may be used at any time.

4. If the evaluated error code specification equals a null vector, the function-name of the right argument is applied as error control for all data-dependent errors.

Resetting Error Control for Data-Dependent Errors

RESETERR resets error control on data-dependent errors. The function is monadic. Argument is error code specification that identifies which errors are to be reset.

Format

RESETERR error code specification

Examples

```
RESETERR ⍳0      Reset all data-dependent errors

RESETERR  4      Reset error code 4
```

Usage Rules

1. The error code specification may be in the form of any APL expression that, evaluated, equals an integer scalar or vector with values between 1 and 29, inclusive.

2. If the evaluated error code specification equals a null vector, all data-dependent errors are reset.

3. The RESETERR function may be used at any time.

Setting Error Control for Deadlock and Non-Data-Dependent Errors

DMSABORT and DMSLOCK specify the name of the function that is to be invoked if deadlock or a non-data-dependent error occurs. The function is monadic. Argument is a text vector containing the name of the function to be invoked if the indicated error occurs.

Format

```
{DMSABORT}
{DMSLOCK } 'function name'
```

Examples

```
DMSABORT 'ABORTFUNC'
DMSLOCK      "        Quote marks signify a null-character vector: Reset deadlock control.
```

Usage Rules

1. The named function must be a nyladic, no-result function. If at the time of occurrence of a deadlock or non-data-dependent error, the named function is defined as other than a nyladic, no-result function, an APL-level error is generated. (See "APL-Level EDMS Errors" below.)

2. If the right argument is a null text vector, control for deadlock or non-data-dependent errors is reset.

3. The DMSABORT and DMSLOCK functions may be used at any time.

Obtaining the EDMS Error Code for an APL-Level Error

DMSERCOD is a nyladic function that results in an integer value representing an APL-level EDMS error as listed in Table F-10 (Appendix F), or in zero if no such error has occurred. Typically, this function will be used in an APL sidetracking procedure. See "APL-Level EDMS Errors", below. The DMSERCOD function may be used at any time.

Checkpointing Database Operations

DMSCHKPT, DMSRLSE, and DMSRECV are nyladic functions used to checkpoint processing or purge the monitor enqueue tables and, optionally, roll back the database to its state at the time of the last DMSRLSE. DMSCHKPT and DMSRLSE are subject to the same usage rules as their standard EDMS counterparts. DMSRECV, being equivalent to EDMS DMSRLSE with recovery, is similarly subject to its usage rules.

Reference Code Conversions

Two functions are available: ECDREF to encode reference codes in the form of integer matrixes containing area, page, and line numbers, and DCDREF to decode the encoded reference code.

Encoding Reference Codes

ECDREF is a monadic function whose argument is a three-element integer vector or an n x 3 integer matrix. The result of the function is an integer vector of encoded reference codes. The length of the vector is the same as the first dimension of matrix arguments, i.e., the number of reference codes constituting the vector will be the same as the number of matrix rows. If the argument is a vector, the result will be a single-element vector. See encode/decode examples below.

Usage Rules

1. A matrix argument may have any number of rows but it must have three columns. The first column is area number, the second column is page number and the third column is line number. If the argument is a vector, it is treated as a 1 x 3 matrix.

2. All area numbers in the argument must be defined in the subschema. The value specified for page number must not exceed 24 bits minus the number of bits required to specify the line number. Line numbers may not exceed the maximum number of lines per page for the given area.

3. The ECDREF function may be used at any time after the DMSSUB call and before a DMSEND call.

Decoding Reference Codes

DCDREF is a monadic function whose argument is an integer scalar or vector of encoded reference codes. Result is an integer matrix whose first dimension (number of rows) is the number of reference codes in the argument, and whose second dimension (number of columns) is three: area numbers, page numbers, and line numbers, in this order.

Usage Rules

1. The values given by LARG ÷ 2*24 must all be numbers of areas defined in the subschema.

2. The DCDREF function may be used at any time after the DMSSUB call and before the DMSEND call.

Examples

```
1.              BREFCODE                        CCB inquiry
    16779010                                        Result
                DCDREF    BREFCODE
        1   7   2
                ECDREF    DCDREF    BREFCODE
    16779010

2.              CURRSET 'SETD'                  Current of set inquiry
    16777473    16778758    16779010    16779558        Result
    (a)         DCDREF    16777473    16778758    16779010    16779558
    1 1 1                                       Result
    1 6 6
    1 7 2
    1 9 38

    (b)         DCDREF  CURRSET 'SETD'          Argument is Current of Set inquiry
    1 1 1                                       Result
    1 6 6
    1 7 2
    1 9 38

3.              ECDREF  DCDREF  CURRSET 'SETD'   Composite argument
    16777473    16778758    16779010    16779558        Result
```

Execution Tracing

DMSTRACE initiates and ENDTRACE terminates the DBM procedural trace which performs the execution tracing. The functions are nyladic and may be used at any time.

APL Command Restrictions

The following APL commands are disallowed while database areas are open and will result in an APL-level EDMS error if any attempt is made to execute them.

LOAD	CONTINUE
CLEAR	SAVE
OFF	

APL-Level EDMS Errors

All EDMS-associated errors detected by the EDMS intrinsic function are handled at the APL level. A special error type exists within APL for EDMS errors. This error (numbered 99) may be sidetracked like any other APL error. If the user chooses to sidetrack on the EDMS error, he may use the DMSERCOD function to determine the nature of the error. Table F-10 in Appendix F details the possible errors and their associated codes.

If the user chooses not to sidetrack on the EDMS error, a message indicating the type of error is printed immediately before the APL-level error is initiated. A list of these messages appears in Table F-10.

Errors in EDMS Functions

Errors detected by the EDMS intrinsic function are reported by APL as having occurred in one of the functions supplied in the DMSFNS workspace. This does not mean that the error was caused by the function that called the intrinsic function; rather it means that the intrinsic call was the point in the APL program at which the error was detected. Usually a domain, rank, or length error that occurs during execution of a function from the DMSFNS workspace is an indication that the EDMS intrinsic is rejecting one of the user's arguments.

Dynamics of EDMS – Level Error Control

Nearly all of the standard EDMS errors are possible under the APL/EDMS interface. These errors are reported in the usual manner, i.e., the error code is placed in the ERR-CODE cell in the CCB. Due to the interpretive nature of the execution of APL programs and the inability to unconditionally transfer control to an arbitrary point outside of the currently executing function, an error control interface has been devised that operates in a manner similar to the APL CATCH debugging facility.

When an EDMS error occurs for which error control has been set, the function named to receive control is tested to insure that it is a nyladic, no-result function. If it is not, an APL-level error occurs. If it is, the function is invoked just as if it had been called explicitly by the user function that called the EDMS function. If the error control function chooses to ignore the error, it may resume execution merely by exiting. Thus a DMSRTRN call is implicit in the fact that the error control function exits.

DMSFNS Workspace

The DMSFNS workspace will normally reside in the DMSLIB account. In addition to the functions described earlier in this chapter, this workspace contains several utility functions that perform various argument-formatting and validation services. The user does not need to be concerned with them beyond being aware of their existence. This category of functions is characterized by a delta-underscore character (⍙) which is both the first and last character in the function name.

The workspace contains three intrinsic functions: ⍙DMS, the EDMS intrinsic; ⍙WM, the workspace-management intrinsic; and, ⍙TE, the text-editing intrinsic. The ⍙TE and ⍙WM intrinsics are used to assist in argument manipulation and to insure that all of the functions in the workspace are independent of the ORIGIN setting.

One APL "group" of functions, named RTRVGRP and associated with retrieval, is contained in the workspace. Thus, the user who is writing a workspace to do only retrieval may include only those functions associated with retrieval

and exclude all those associated with updating. Also, the user may selectively exclude entry-point functions that are not used. For example, a user who is writing a workspace to generate a report that does not require the FINDS procedure, may copy the RTRVGRP out of DMSFNS and then erase the FINDS function.

Subschema File

The APL/EDMS interface requires a subschema that (1) was generated by DMSFDP, version C00 or later, and (2) contains a name table. If either condition is not met, an APL-level EDMS error is reported.

Area File DCBs

Since database area files are accessed via the APL file I/O DCBs, APL/EDMS users are limited to a total of eight concurrently open files and database areas. No assignment of the associated DCBs is necessary or possible. Execution of an EDMS open function initializes the DCBs with file name, account number and password. Files on private disk packs may be accessed with the DMSPKSN call. Area files use file tie numbers that are the negative of the area number (i.e., area number 16 has file tie number -16). Use of the APL I/O primitive operator 19 will thus identify which DCBs are currently in use by EDMS.

The user should note that the EDMS intrinsic allocates a DCB during each open call and deallocates all DCBs when the last database area is closed. This means that database areas may have DCBs tied up even though they are not actually in use.

Journal and Statistics DCBs

The permanent journal is written through DCB F:JRNL. It will be written if the schema DDL specifies journaling for an area and that area is opened for update. The EDMS intrinsic function uses a journal default assignment to file name 'JRNL-ID', where ID is the system job ID. This file is created in the run account. If the database is opened several times in a single APL/EDMS session, the EDMS intrinsic will extend the file on each open. The user may set DCB F:JRNL to labeled tape or to a different file name prior to entering the processor.

If the schema DDL specifies that statistics are to be created on any group or set in the subschema, the EDMS intrinsic function writes the statistics through the F:STAT DCB. The file name is defaulted to 'STAT-ID', where ID is the system job ID. The user may set DCB F:STAT to labeled tape or to a different file name prior to entering the processor.

APPENDIX A. SCHEMA FILE

The schema file is itself an EDMS database of only one area. The "data" in this database is information about the user database that is defined by the schema DDL. The user's database is defined in the schema in terms of its area, group, item, and set components. The schema also contains the subschema names of all subschemas that have been generated using the schema. (Subschema information does not exist in the schema when it is initially created.) Figure A-1 illustrates the schema database relationships. The groups and sets are explained below. Table A-1 contains explanations of the items. Figure A-2 shows the schema Data Definition Language used to define the schema database.

There is only one occurrence of the group SCHEMAHD. It is stored on page 1, line 1 of the file and is the basic entry point to the schema database.

Linked to the SCHEMAHD occurrence are

1. One occurrence of the ASOWNER group for each set defined in the user's database (schema database set W).

2. One occurrence of the PASSWORD group for each password defined for the user's database (schema set A).

3. One occurrence of the SSCHEM group for each subschema defined (set B). These occurrences are added by the FDP when subschemas are generated.

4. One occurrence of AREAGP group for each area in the user's database (set C).

If an area contains an indexed-sequential group, its AREAGP occurrence has associated INDX group occurrences describing the several significant page ranges in the area (set F). The significant page ranges are the range specified for the indexed group, the overflow range, and the range of pages used for each level of indexing.

An AREAGP occurrence also serves as an entry point (through set E) to information on all the groups in the area, including information on the sets of which the groups are defined as owners and members.

The UNIT group occurrences contain the basic information on user's groups (size, location mode, etc.). Linked to each UNIT group occurrence (through set H) is one occurrence of the ELEMENT group for each item defined for the group. Also linked to each UNIT group is an occurrence of the ASOWNER group for each set of which the referent group is an owner (set J), and an occurrence of the ASMEMBER group for each set of which the reference group is a member (set I).

Each ASMEMBER occurrence is linked (set M) to an ASOWNER occurrence for the referent set. (An ASMEMBER and an ASOWNER occurrence that are linked together are, of course, linked to separate UNIT occurrences.) An ASMEMBER occurrence may be linked to one or more ASCONTROL group occurrences through set N. The ASCONTROL occurrences associate an ASMEMBER occurrence with the ELEMENT occurrence(s) that describe the item(s) defined as sort keys for the set.

An ASMEMBER occurrence may be associated indirectly with an ELEMENT occurrence by means of an ALIAS group occurrence using sets O and U. Each UNIT group occurrence is linked to one or more GROUPRET group occurrences (set G) and may be linked to one GSTATS group occurrence.

There is an occurrence of a NAMEGP group for each item, group, or set defined for the user's database. There is also a NAMEGP occurrence for each alias name specified. Each NAMEGP occurrence is linked to SNAMLINK, INAMLINK, GNAMLINK, and/or ALIAS occurrence.

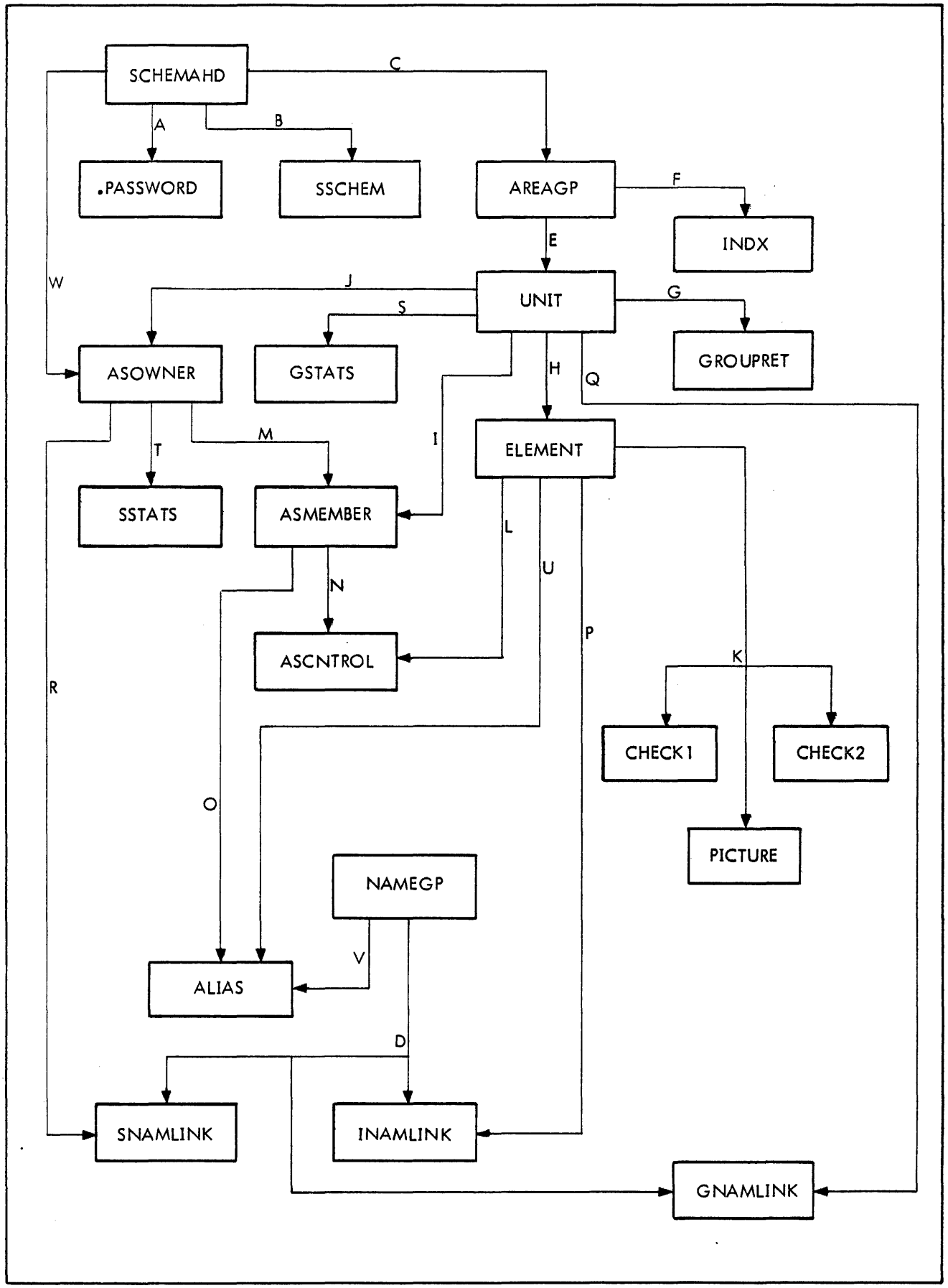


Figure A-1. Schema Database Diagram

Table A-1. Schema Items

Group	Item	Explanation
(1) AREAGP	NAMESIZE	Number of characters in area name.
	AREANAME	User-supplied name.
	AREANO	User-assigned number.
	INVPERCT	Inventory percent assigned by user– 50% minimum.
	NBROFLIN	Lines per data page: 1 implies 16, 2 implies 32, 3 implies 64, 4 implies 128, 5 implies 256.
	CHECKSUM	0 – no checksum on data pages; 1 – checksum.
	FILPERCT	Percentage of page DBM is to use when area is created.
	JOURNAL	0 – no journal; 1 – journal.
	ENCIPHER	0 – do not cipher data pages; 1 – cipher pages.
	INDEXED	0 – area not indexed; 1 – area indexed.
	AOWNER	0 – area not owner of any sets; 1 – area owns sets.
	AREAFIL1	Unused.
	DATAPGES	Number of data pages.
	PAGESIZE	Number of words per data page (currently fixed at 512).
	KEYSIZE	Size of indexed key in bytes if area is indexed.
	RETUSERS	Number of retrieve users.
	UPDUSERS	Number of update users.
	PAGEIO	Number of physical page I/Os.
	GRPSACSD	Number of groups accessed.
	GRPSINSD	Number of groups inserted.
	GRPSDLTD	Number of groups deleted.
	AREAFIL2	Unused.
(2) UNIT	GROUPNO	User-supplied number.
	LOCATMOD	Location Mode: 1 for direct; 2 for indexed; 3 for calc; 4 for calcdup; 5 for via.
	INVTITEM	0 – no inverted items in group; 1 – inverted items.
	GRPRLOCK	User-supplied retrieve lock (maximum value = 255).
	GRPULOCK	User-supplied update lock (maximum value = 255).
	STRGESET	0 – no storage set; 1 – storage set specified.
	SECINDEX	0 – not secondary index; 1 – group is secondary index.
	NUMKEYS	Number of calc, index of sort key items (0-7).
	DEFRGE	0 – user supplied page range; 1 – default.
	GRPFILL1	Unused.
	GRPSIZE	Size of group in bytes.
	BEGPGRGE ENDPGRGE	Page range for group.
	PRIMVALU	Prime number for hash of calc groups.
	GRPFIL2	Unused.

Table A-1. Schema Items (cont.)

Group	Item	Explanation
(3) ASOWNER	SETFILL1	Unused.
	SETNO	Sequential number for set.
	OPSTNEXT	Relative byte position of set NEXT pointer.
	OPSTNPRI	Relative byte position of set PRIOR pointer.
	SETFILL2	Unused.
(4) ASMEMBER	ORDER	0 – implies last; 1 – prior; 4 – sorted; 8 – first; 9 – next.
	GRPNOKY	Group number as sort key; 0 implies not applicable; 1 – ignore; 2 – major; 3 – minor.
	DUPSIND	Duplicates indicated: 0 implies not allowed; 1 – first; 2 – last.
	OPTIONAL	0 for membership not optional; 1 – membership is optional.
	AUTOMANL	0 for membership is automatic; 1 – membership is manual.
	PRIMARY	0 – not primary set for group; 1 – set is primary.
	STORAG	0 – not storage set for group; 1 – set is storage set.
	SELOWNER	0 – owner selection is unique; 1 – owner selection is current.
	OWNERNO	Unused.
	MEMBFIL1	Unused.
	MPSTNEXT	Relative byte position of set NEXT pointer.
	MPSTNPRI	Relative byte position of set PRIOR pointer.
	PSTNHEAD	Relative byte position of set HEAD pointer.
	MEMBFIL2	Unused.
(5) ELEMENT	ITEMTYPE	0 – signed numeric; 1 – alphanumeric; 2 – numeric; 3 – alphabetic; 4 – binary; 5 – floating-point short; 6 – floating-point long; 7 – packed decimal.
	LEVELNBR	Will not be used in the current FDP.
	OCCURCNT	Number of occurrences of this item.
	ITMRLOCK	User-supplied retrieve lock (maximum value = 255).
	ITMULOCK	User-supplied update lock (maximum value = 255).
	INVTDNO	Number of secondary index group.
	DATAVLID	Data validation type: 0 implies none; 1 – picture; 2 – range; 3 – both.
	CONTROL	0 – item not calc or index control; 1 – item is control.
	DEFPIC	1 – Defaults picture supplied for packed decimal item.
	ITEMFIL1	Unused.
	ITEMPSTN	Relative byte position of item in group.
	ITEMSIZE	Size of item in bytes.
	ITEMFIL2	Unused.
	ITEMFIL3	Unused.
	ITEMFIL4	Unused.

Table A-1. Schema Items (cont.)

Group	Item	Explanation
(6) ASCNTROL	MATCHIND	Type of sort match: 0 for equal; 1 for range.
	CTRLTYPE	Sequence Control Type: 1 for ascending; 2 for descending.
	CTRLFIL1	Unused.
(7) SCHEMAHD	COPYPSWD	EXTRACT privacy lock.
	ALTRPSWD	ALTER privacy lock (not currently used).
	PTRSIZE	Size of set pointers in bytes.
	SCHFIL1	Unused.
	SCHDATIM	Date and time when schema was created.
	SCHESIZE	Size of schema in pages.
	NUMPSWDS	Count of password groups.
	NUMOWNRS	Count of ASOWNER groups.
	NUMMBRS	Count of ASMEMBER groups.
(8) PASSWORD	PASSWORD	User-supplied database access password.
	RETKEYS	Retrieve keys for this password – one bit for each value up to 255.
	UPDKEYS	Update keys for this password – one bit for each value up to 255.
(9) SSCHEM	SUBSNAME	Subschema name.
	ACCTNBR	Account number under which subschema was created.
	SUBDATE	Date created (halfword binary year and halfword binary julian day).
	SUBSTIME	Time created: byte 0 = hour (0-23); byte 1 = minutes (0-59); byte 2 = second (0-59); byte 3 = hundredths of a second (0-99).
(10) INDX	BEGPGNBR ENDPGNBR	Beginning and ending page numbers which together define an INDX overflow range or index level.
	DEFNTYPE	Type of definition: 0 for overflow range; 1 for index level.
	INDXLEVL	Index level number (0 implies indexed data group page range).
	INDXFIL1	Unused.
(11) PICTURE	PICTCNT	Number of characters in picture.
	ITEMPICT	User-supplied picture for item
	SCALE	Scaling factor for picture.
	PICFIL1	Unused.
(12) CHECK1	LOWLIT1 HILIT1	Low and high literals for data validation (CHECK clause) of binary and floating-point short items.
	CK1FIL1	Unused.

Table A-1. Schema Items (cont.)

Group	Item	Explanation
(13) CHECK2	LOWLIT2 HILIT2	Low and high literals for data validation (CHECK clause) of floating-point long, packed decimal, and EBCDIC items. Floating-point long literals will be in the first two words of each item. Packed decimal will always be 16 bytes. EBCDIC literals will be left-justified in each item.
	CK2FIL1	Unused.
(14) ALIAS	(No items)	
(15) GROUPRET	DATNAME	Name of retrieval item or set, or sort key.
	RTVLTYPE	Retrieval type: 1 implies index name; 2 – calc item name; 3 – via setname; 4 – storage setname; 5 – sort key name.
	GRFILL1	Unused.
(16) NAMEGP	NAMEVALU	User-supplied name for set, group or item.
	PRIMNAME	Not used in current version.
	NAMETYPE	1 implies setname; 2 – group name; 3 – item name; 0 – none.
	DUPNAME	0 if no other item in schema has this name; 1 – duplicates exist.
	NAMFIL1	Unused.
(17) INAMLINK	(No items)	
(18) GNAMLINK	(No items)	
(19) SNAMLINK	(No items)	
(20) GSTATS		(Reserved for future implementation.)
	NBRACSD	Number of group occurrences accessed.
	NBRINSD	Number of group occurrences inserted.
	NBRDLTD	Number of group occurrences deleted.
(21) SSTATS		(Reserved for future implementation.)
	HEADACCS	Number of head accesses through this set.
	NEXTACCS	Number of next accesses through this set.
	PRIRACCS	Number of prior accesses through this set.

```
SCHEMA IS SCHEMASCHEMA.

AREA IS SCHEBASE CONTAINS 1 PAGES
     ; NUMBER IS 1
     ; ENCIPHERING IS NOT REQUIRED
     ; CHECKSUM IS REQUIRED
     ; JOURNAL IS NOT REQUIRED
     .

GROUP IS AREAGP
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS CALC USING AREANAME DUPLICATES NOT ALLOWED
     ; NUMBER IS 1
     .

     NAMESIZE; PIC X.
     AREANAME; PIC X(30).
     AREANO ; PIC X.
     INVPERCT; PIC 99.
     NBROFLIN; PIC 9.
     CHECKSUM; PIC 9.
     FILPERCT; PIC 99.
     JOURNAL; PIC 9.
     ENCIPHER; PIC 9.
     INDEXIND; PIC 9.
     AOWNER; PIC 9.
     AREAFIL1; TYPE IS BINARY.
     DATAPGES; TYPE IS BINARY.
     PAGESIZE; TYPE IS BINARY.
     KEYSIZE; TYPE IS BINARY.
     RETUSERS; TYPE IS BINARY.
     UPDUSERS; TYPE IS BINARY.
     PAGEIO; TYPE IS BINARY.
     GRPSACSD; TYPE IS BINARY.
     GRPSINSD; TYPE IS BINARY.
     GRPSDLTD; TYPE IS BINARY.
     AREAFIL2; TYPE IS BINARY.

GROUP IS UNIT
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS CALC USING GROUPNO DUPLICATES NOT ALLOWED
     ; NUMBER IS 2
     .

     GROUPNO; PIC 9(4).
     LOCATMOD; PIC 9.
     INVTITEM; PIC 9.
     GRPRLOCK; PIC 999.
     GRPULOCK; PIC 999.
     STRGESET; PIC 9.
     SECINDEX; PIC 9.
     NUMKEYS; PIC X.
     DEFRGE; PIC 9.
     GRPFIL1; TYPE IS BINARY.
     GRPSIZE;  TYPE IS BINARY.
     BEGPGRGE;  TYPE IS BINARY.
     ENDPGRGE;  TYPE IS BINARY.
     PRIMVALU;  TYPE IS BINARY.
     GRPFIL2; TYPE IS BINARY.

GROUP IS ASOWNER
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS VIA OWNERSET
     ; NUMBER IS 3
     .
```

Figure A-2. Schema DDL for Schema

```
    SETFILL1; TYPE IS BINARY.
    SETNO; PIC 9(4).
    OPSTNEXT; TYPE IS BINARY.
    OPSTNPRI; TYPE IS BINARY.
    SETFILL2; TYPE IS BINARY.

GROUP IS ASMEMBER
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA MEMBRSET
    ; NUMBER IS 4
    .

    ORDER; PIC 9.
    GRPNOKY; PIC 9.
    DUPSIND; PIC 9.
    OPTIONAL; PIC 9.
    AUTOMANL; PIC 9.
    PRIMARY; PIC 9.
    STORAG; PIC 9.
    SELOWNER; PIC 9.
    OWNRNO; PIC 9(4).
    MEMBFIL1; TYPE IS BINARY.
    MPSTNEXT; TYPE IS BINARY.
    MPSTNPRI; TYPE IS BINARY.
    PSTNHEAD; TYPE IS BINARY.
    MEMBFIL2; TYPE IS BINARY.

GROUP IS ELEMENT
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA ITEMSET
    ; NUMBER IS 5
    .

    ITEMTYPE; PIC 9.
    LEVELNBR; PIC 999.
    OCCURCNT; PIC 999.
    ITMRLOCK; PIC 999.
    ITMULOCK; PIC 999.
    INVTDNO; PIC 999.
    DATAVLID; PIC 9.
    CONTROL; PIC 9.
    DEFPIC; PIC 9.
    ITEMFIL1; TYPE IS BINARY.
    ITEMPSTN; TYPE IS BINARY.
    ITEMSIZE; TYPE IS BINARY.
    ITEMFIL2; TYPE IS BINARY.
    ITEMFIL3; TYPE IS BINARY.
    ITEMFIL4; TYPE IS BINARY.
GROUP IS ASCNTROL
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA CTRLSET
    ; NUMBER IS 6
    .

    MATCHIND; PIC 9.
    CTRLTYPE; PIC 9.
    CTRLFIL1; PIC 99.

GROUP IS SCHEMAHD
    ; WITHIN SCHEBASE, RANGE IS PAGE 1 THRU PAGE 1
    ; LOCATION MODE IS DIRECT
    ; NUMBER IS 7.
```

Figure A-2. Schema DDL for Schema (cont.)

```
     COPYPSWD; PIC X(8).
     ALTRPSWD; PIC X(8).
     PTRSIZE; PIC 9.
     SCHFIL1; PIC XXX.
     SCHDATIM; PIC X(20).
     SCHESIZE; TYPE IS BINARY.
     NUMPSWDS; TYPE IS BINARY.
     NUMOWNRS; TYPE IS BINARY.
     NUMMBRS; TYPE IS BINARY.

GROUP IS PASSWORD
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS VIA PASSWSET
     ; NUMBER IS 8
     .

     PASSWRD; PIC X(8).
     RETKEYS; PIC X(32).
     UPDKEYS; PIC X(32).

GROUP IS SSCHEM
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS VIA SSCHMSET
     ; NUMBER IS 9
     .

     SUBSNAME; PIC X(30).
     ACCTNBR; PIC X(8).
     SUBSDATE; TYPE IS BINARY.
     SUBSTIME; TYPE IS BINARY.

GROUP IS INDX
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS VIA INDEXSET
     ; NUMBER IS 10
     .

     BEGPGNBR; TYPE IS BINARY.
     ENDPGNBR; TYPE IS BINARY.
     DEFNTYPE; PIC 9.
     INDXLEVL; PIC 9.
     INDXFIL1; PIC 99.

GROUP IS PICTURE
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS VIA DESCPSET
     ; NUMBER IS 11
     .

     PICTCNT; PIC X.
     ITEMPICT; PIC X(30).
     SCALE; PIC X.
     PICFILL1; TYPE IS BINARY.

GROUP IS CHECK1
     ; WITHIN SCHEBASE
     ; LOCATION MODE IS VIA DESCPSET
     ; NUMBER IS 12
     .

     LOWLIT1; TYPE IS BINARY.
     HILIT1; TYPE IS BINARY.
     CK1FIL1; TYPE IS BINARY.
```

Figure A-2. Schema DDL for Schema (cont.)

```
GROUP IS CHECK2
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA DESCRSET
    ; NUMBER IS 13
    .

    LOWLIT2; PIC X(16).
    HILIT2; PIC X(16).
    CK2FIL1; TYPE IS BINARY.

GROUP IS ALIAS
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA ALIASSET
    ; NUMBER IS 14
    .

GROUP IS GROUPRET
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA GRPRET
    ; NUMBER IS 15
    .

    DATNAME; PIC X(30).
    RTVLTYPE; PIC 9.
    GRFILL1; PIC 9.

GROUP IS NAMEGP
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS CALC USING NAMEVALU DUPLICATES NOT ALLOWED
    ; NUMBER IS 16
    .

    NAMEVALU; PIC X(30).
    PRIMNAME; PIC 9.
    NAMETYPE; PIC 9.
    DUPNAME; PIC 9.
    NAMFIL1; PIC 9(3).

GROUP IS INAMLINK
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA INAMESET
    ; NUMBER IS 17
    .

GROUP IS GNAMLINK
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA GNAMESET
    ; NUMBER IS 18
    .

GROUP IS SNAMLINK
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA SNAMESET
    ; NUMBER IS -19
    .

GROUP IS GSTATS
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA GSTATSET
    ; NUMBER IS 20
    .

    NBRACSD; TYPE IS BINARY.
    NBRINSD; TYPE IS BINARY.
    NBRDLTD; TYPE IS BINARY.
```

Figure A-2. Schema DDL for Schema (cont.)

```
GROUP IS SSTATS
    ; WITHIN SCHEBASE
    ; LOCATION MODE IS VIA SSTATSET
    ; NUMBER IS 21
    .

    HEADACCS; TYPE IS BINARY.
    NEXTACCS; TYPE IS BINARY.
    PRIRACCS; TYPE IS BINARY.




/*
         S E T S
                              */
SET IS PASSWSET
    ; OWNER IS SCHEMAHD
    ; ORDER IS SORTED
    .

    MEMBER IS PASSWORD
         ; INCLUSION IS AUTOMATIC
         ; SELECTION IS CURRENT
         ; ASCENDING KEY IS PASSWRD DUPLICATES NOT ALLOWED
         .

SET IS SSCHEMSET
    ; OWNER IS SCHEMAHD
    ; ORDER IS SORTED
    .

    MEMBER IS SSCHEM
         ; INCLUSION IS AUTOMATIC
         ; SELECTION IS CURRENT
         ; ASCENDING KEY IS SUBSNAME DUPLICATES NOT ALLOWED
         .

SET IS AREASET
    ; OWNER IS SCHEMAHD
    ; ORDER IS LAST
    .

    MEMBER IS AREAGP
         ; INCLUSION IS AUTOMATIC
         ; SELECTION IS CURRENT
         .

SET IS NAMESET
    ; OWNER IS NAMEGP
    ; ORDER IS FIRST
    .

    MEMBER IS INAMLINK
         ; INCLUSION IS AUTOMATIC
         ; SELECTION IS CURRENT
         ; LINKED TO OWNER
         .
```

Figure A-2. Schema DDL for Schema (cont.)

```
     MEMBER IS GNAMLINK
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

     MEMBER IS SNAMLINK
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS GROUPC
     ; OWNER IS AREAGP
     ; ORDER IS LAST
     .

     MEMBER IS UNIT
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS INDEXSET
     ; OWNER IS AREAGP
     ; ORDER IS LAST
     .

     MEMBER IS INDX
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .

SET IS GRPRET
     ; OWNER IS UNIT
     ; ORDER IS LAST
     .

     MEMBER IS GROUPRET
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .

SET IS ITEMSET
     ; OWNER IS UNIT
     ; ORDER IS LAST
     .

     MEMBER IS ELEMENT
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS MEMBRSET
     ; OWNER IS UNIT
     ; ORDER IS LAST
     .

     MEMBER IS ASMEMBER
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .
```

Figure A-2. Schema DDL for Schema (cont.)

```
SET IS OWNERSET
     ; OWNER IS UNIT
     ; ORDER IS LAST
     .

     MEMBER IS ASOWNER
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS DESCPSET
     ; OWNER IS ELEMENT
     ; ORDER IS LAST
     .

     MEMBER IS PICTURE
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .
     MEMBER IS CHECK1
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .

     MEMBER IS CHECK2
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .

SET IS MODFYSET
     ; OWNER IS ELEMENT
     ; ORDER IS LAST
     .

     MEMBER IS ASCNTROL
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS SETLINK
     ; OWNER IS ASOWNER
     ; ORDER IS LAST
     .

     MEMBER IS ASMEMBER
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS CTRLSET
     ; OWNER IS ASMEMBER
     ; ORDER IS LAST
     .

     MEMBER IS ASCNTROL
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS ALIASSET
     ; OWNER IS ASMEMBER
```

Figure A-2. Schema DDL for Schema (cont.)

```
     ; ORDER IS LAST
     .

     MEMBER IS ALIAS
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS INAMESET
     ; OWNER IS ELEMENT
     ; ORDER IS LAST
     .

     MEMBER IS INAMLINK
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS GNAMESET
     ; OWNER IS UNIT
     ; ORDER IS LAST
     .

     MEMBER IS GNAMLINK
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS SNAMESET
     ; OWNER IS ASOWNER
     ; ORDER IS LAST
     .

     MEMBER IS SNAMLINK
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS GSTATSET
     ; OWNER IS UNIT
     ; ORDER IS FIRST
     .

     MEMBER IS GSTATS
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .

SET IS SSTATSET
     ; OWNER IS ASOWNER
     ; ORDER IS FIRST
     .

     MEMBER IS SSTATS
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .
```

Figure A-2. Schema DDL for Schema (cont.)

```
SET IS MKALTSET
     ; OWNER IS ELEMENT
     ; ORDER IS LAST
     .

     MEMBER IS ALIAS
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS ALNAMSET
     ; OWNER IS NAMEGP
     ; ORDER IS LAST
     .

     MEMBER IS ALIAS
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          ; LINKED TO OWNER
          .

SET IS HDRSET
     ; OWNER IS SCHEMAHD
     ; ORDER IS LAST
     .

     MEMBER IS ASOWNER
          ; INCLUSION IS AUTOMATIC
          ; SELECTION IS CURRENT
          .

END.
```

Figure A-2. Schema DDL for Schema (cont.)

APPENDIX B. SUBSCHEMA FILE

The subschema file contains a control block, a list structure that defines all or a part of a database for the Database Manager (DBM) and an optional block of name table entries. Figure B-1 illustrates the relationship among the different categories of data within the list structure used by the DBM.

The list structure contains encoded information on the structure of the database to guide the DBM in its interpretive execution of user's procedural accesses to the database. The list structure also contains a layout of the user's working storage that will exist in every program using this subschema when processing in the database. A complete layout of the list structure is included in Figures B-2 through B-12. Figure B-13 shows the format of the entries in the optional name-table. Figure B-14 shows the subschema file directory block format.

Except for the PASSWORD definition, all the values for LINK NEXT and LINK HEAD in the definitions are offsets from the beginning of the subschema, word 0 of the subschema definition. These values are translated to actual core locations when the subschema is read into core by the DBM. PASSWORD LINK NEXT in the subschema definition refers to a block number of the first block of passwords; PASSWORD LINK NEXT in the PASSWORD definition is nonzero for all but the last definition in the password list.

Subschema links roughly correspond to schema set pointers, though the subschema is not a database.

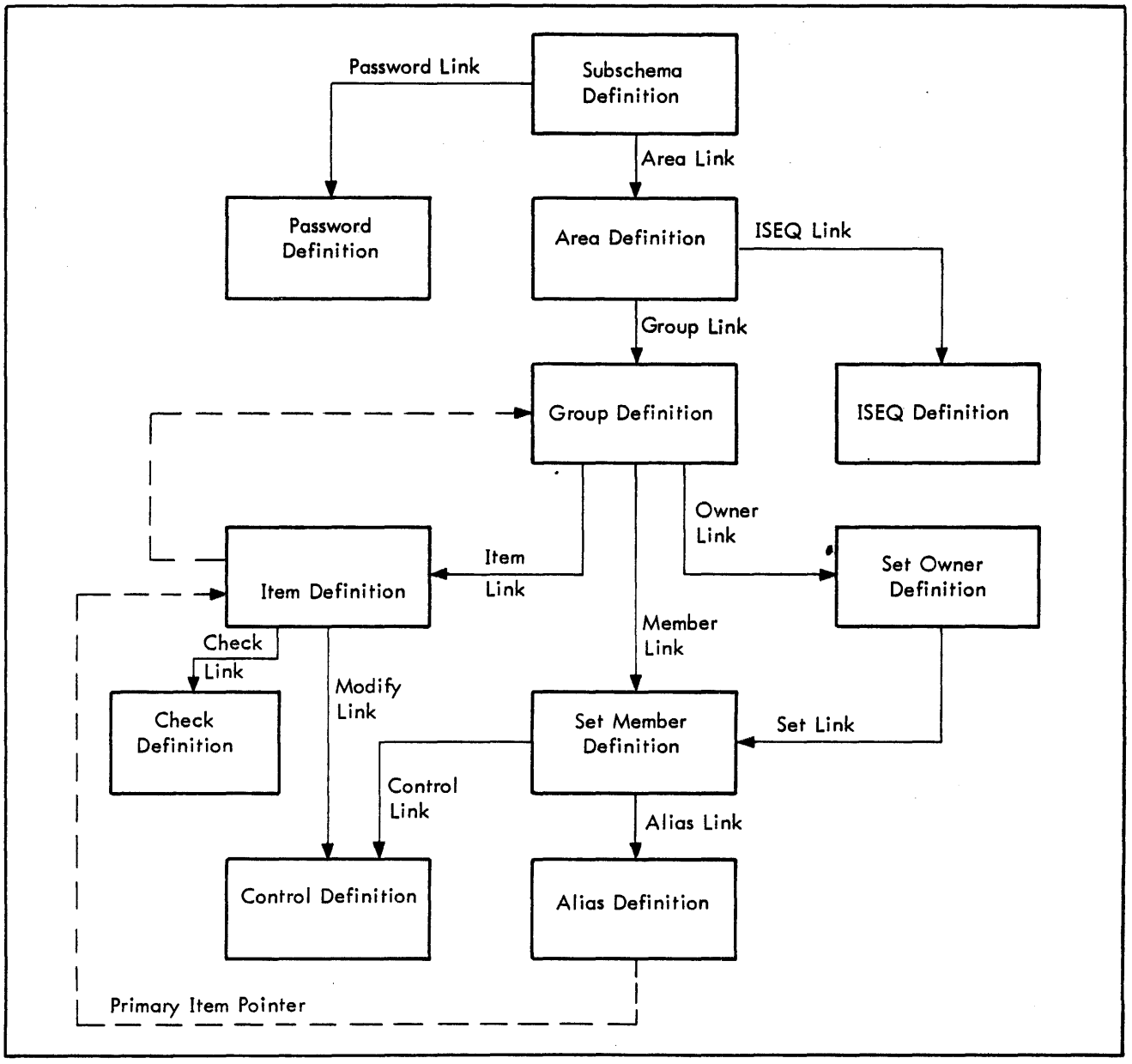
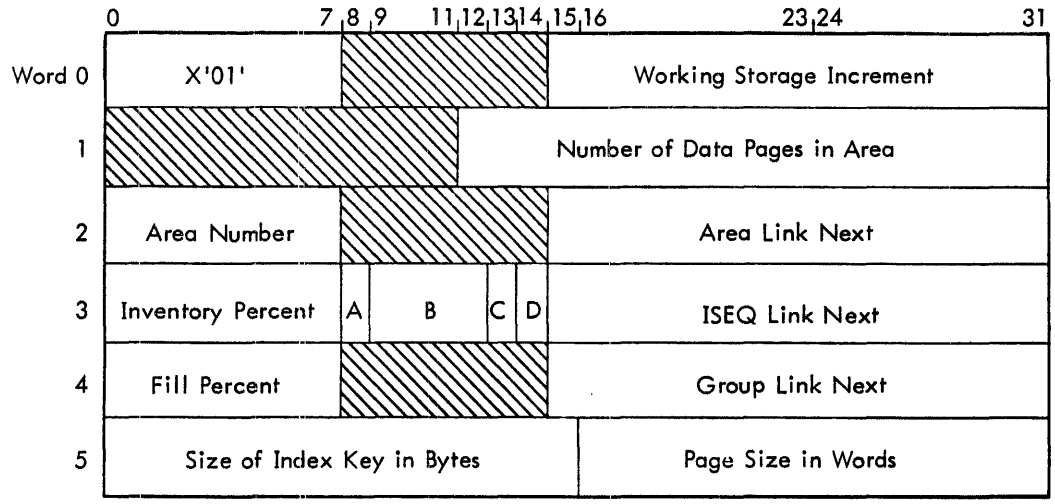


Figure B-1. Subschema Definition Structure



Words 6 through 13 contain the area name in TEXTC format.

where

A = 1 if area has checksums.

B = size of data-page line numbers in bits (4, 5, 6, 7, or 8).

C = 1 if data pages are to be enciphered.

D = 1 if area is to be journaled.

Figure B-2. Area Definition

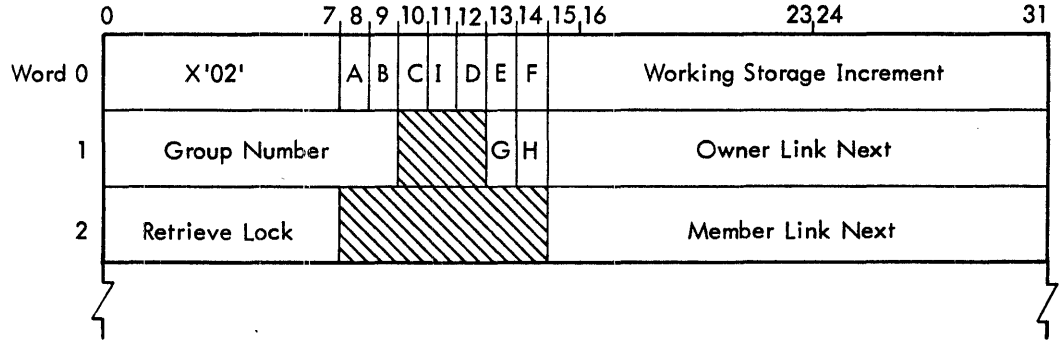
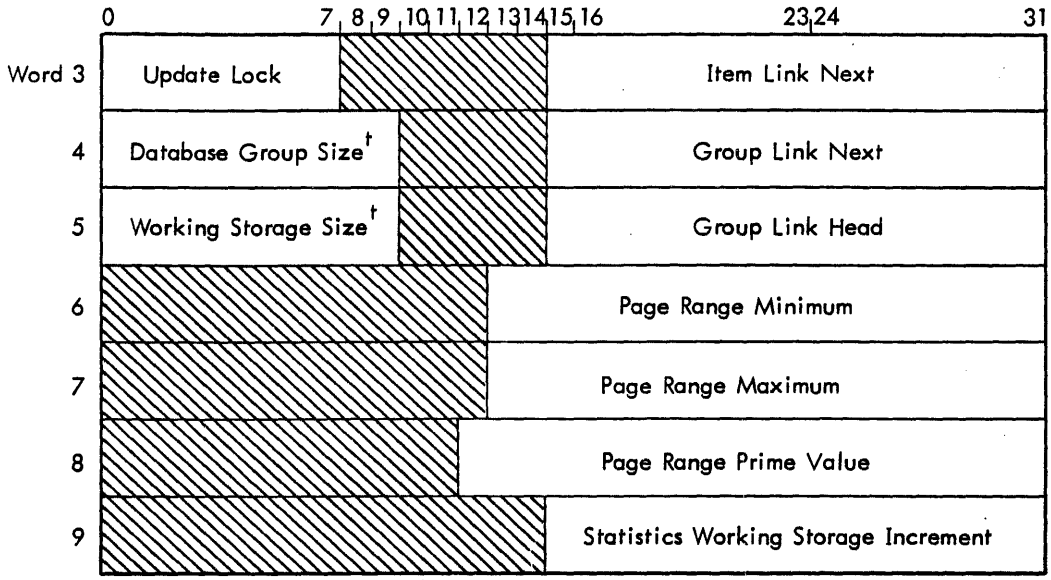


Figure B-3. Group Definition



Words 6 and 7 are optional and present only if bit D is set. Word 8 is optional and present only if bit C is set. Word 9 is optional and present only if bit H is set.

where

A = 1 if group is stored relative to a storage set.

B = 1 if this is a direct group.

C = 1 if this is a calc group.

D = 1 if page range is present.

E = 1 if group has any inverted items.

F = 1 if this is an indexed group.

G = 1 if group cannot be stored because of missing items, sets, or secondary indexes.

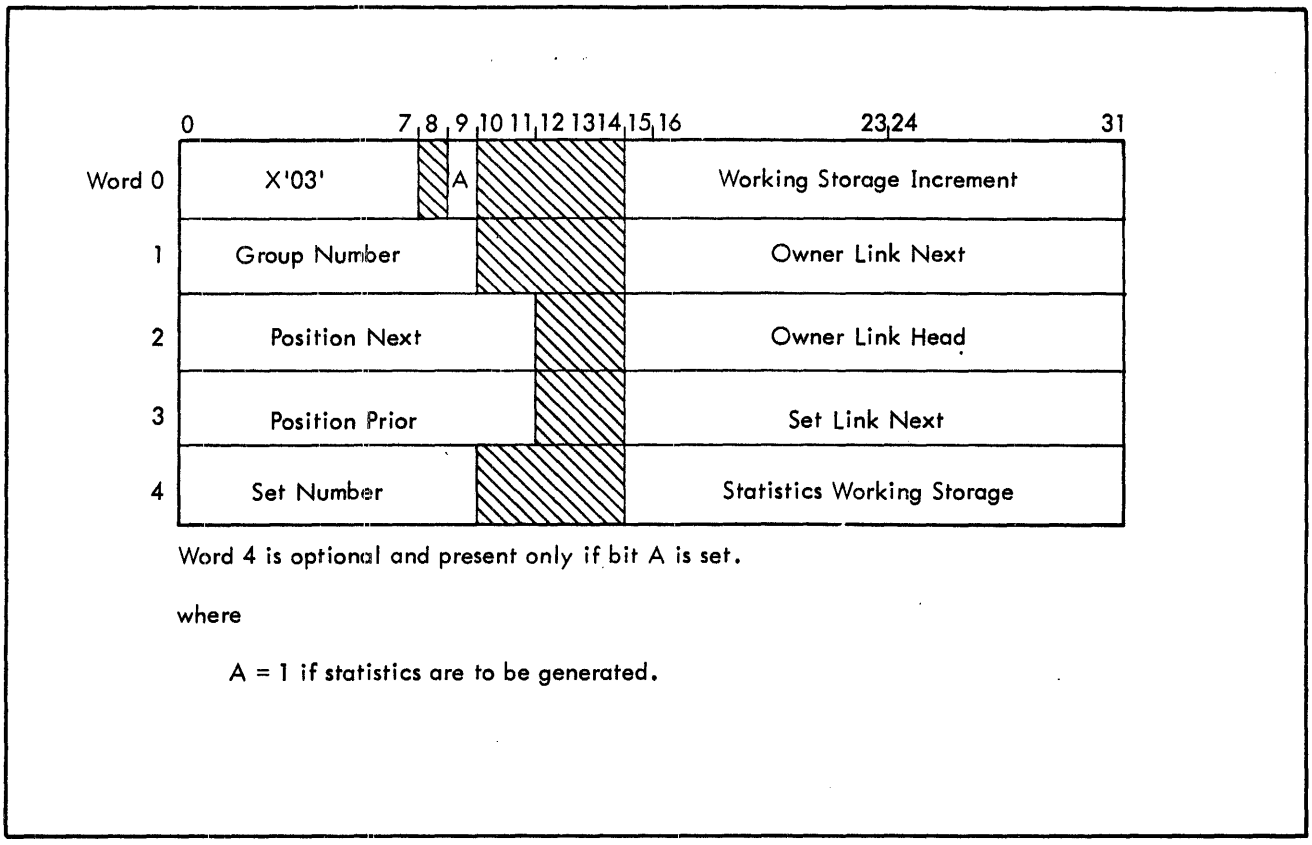
H = 1 if statistics shall be generated for the group.

I = 1 if group cannot be deleted because of missing sets or secondary indexes.

---

†In words.

Figure B-3. Group Definition (cont.)



Word 4 is optional and present only if bit A is set.

where

A = 1 if statistics are to be generated.

Figure B-4. Owner Definition

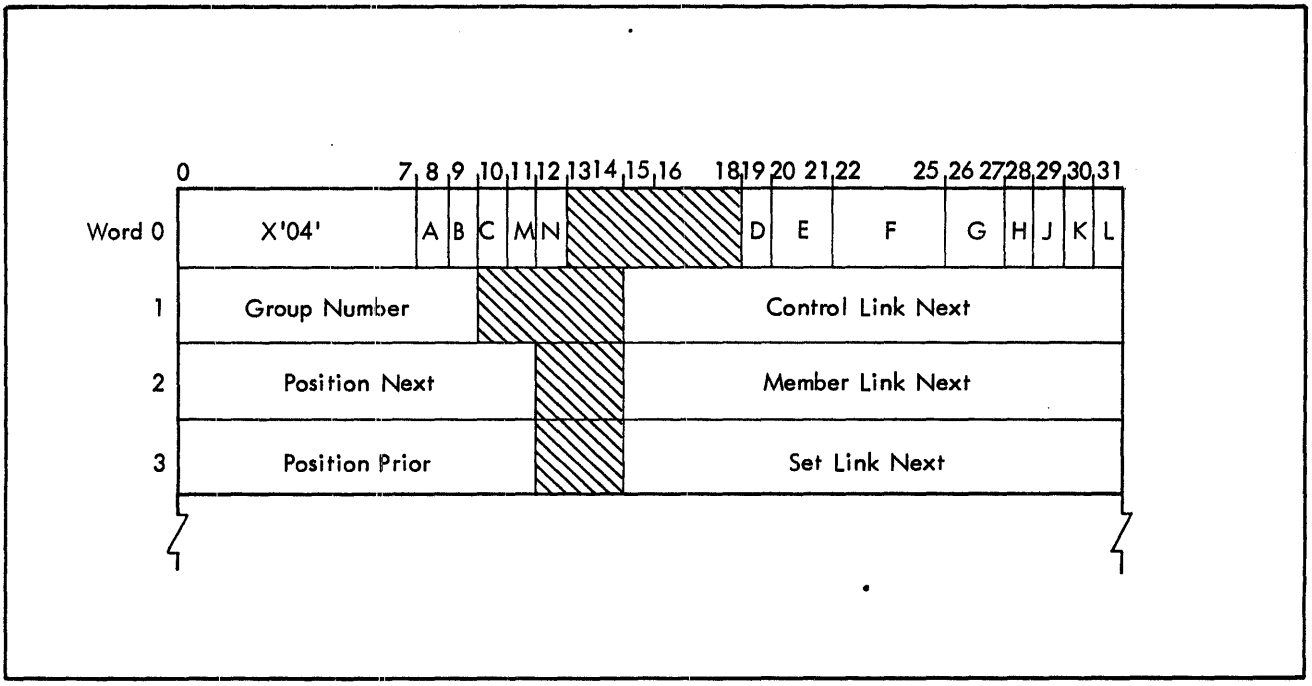
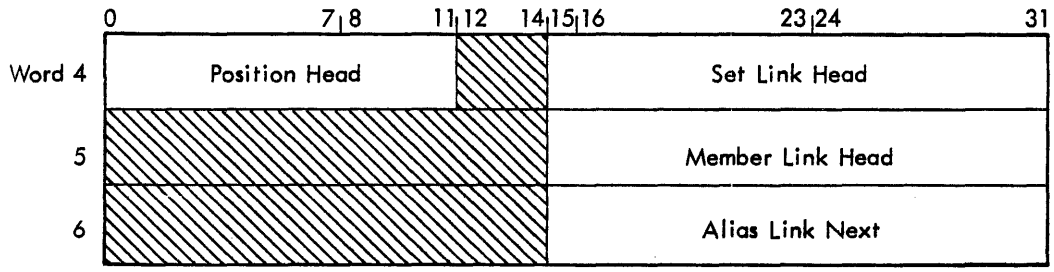


Figure B-5. Member Definition



Word 6 is optional; it is present only if bit N is set and it is used only if bit A is set.

where

A = 1 if there are any aliases defined for the set.

B = 1 if member is optional.

C = 1 if member is manual.

D = 1 if PAGESET member.

E = 01 if group number is major sort key; = 10 if minor.

F indicates set order: 0000 implies last; 0001 – prior; 0100 – sorted; 1000 – first; 1001 – next; 0110 – sorted by group number.

G = 01 implies duplicates first; 10 – duplicates last; 00 – duplicates not allowed.

H = 1 if CALCSET member.

J = 1 if selection is current; = 0 if location mode of owner.

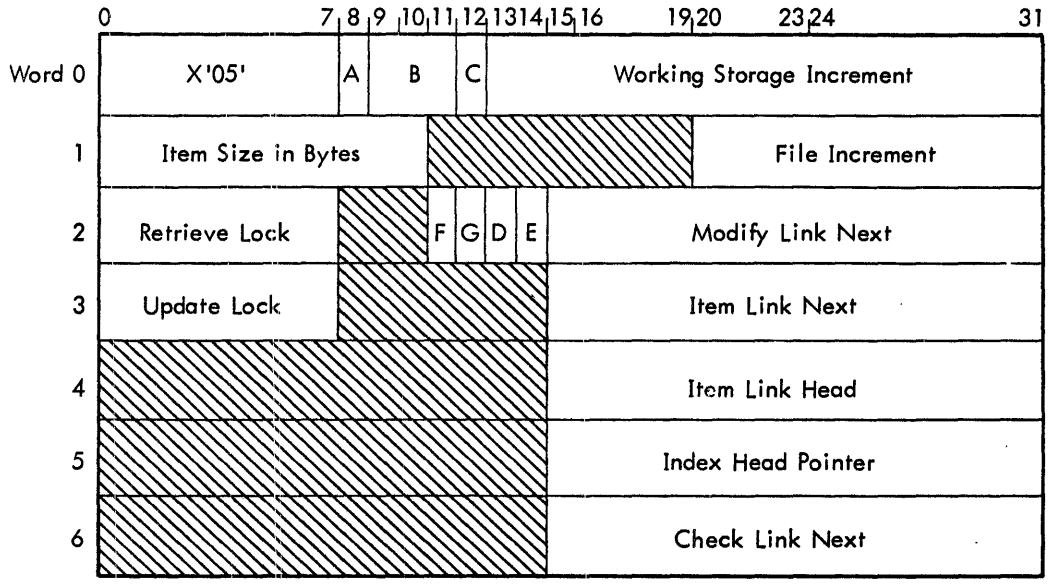
K = 1 if this is storage set for group.

L = 1 if this is prime retrieval set.

M = 1 if control items are omitted.

N = 1 if definition is seven words; = 0 if six words.

Figure B-5. Member Definition (cont.)



Word 5 is optional and present only if bit C is set and bit F is reset. Word 6 is optional and present only if bit D and/or bit E is set.

where

A = 1 if this is control item (calc, index, or sort key on via set).

B indicates item type: 0 implies signed number; 1 – alphanumeric; 2 – numeric; 3 – alphabetic; 4 – binary; 5 – floating short; 6 – floating long; 7 – packed.

C = 1 if item is inverted.

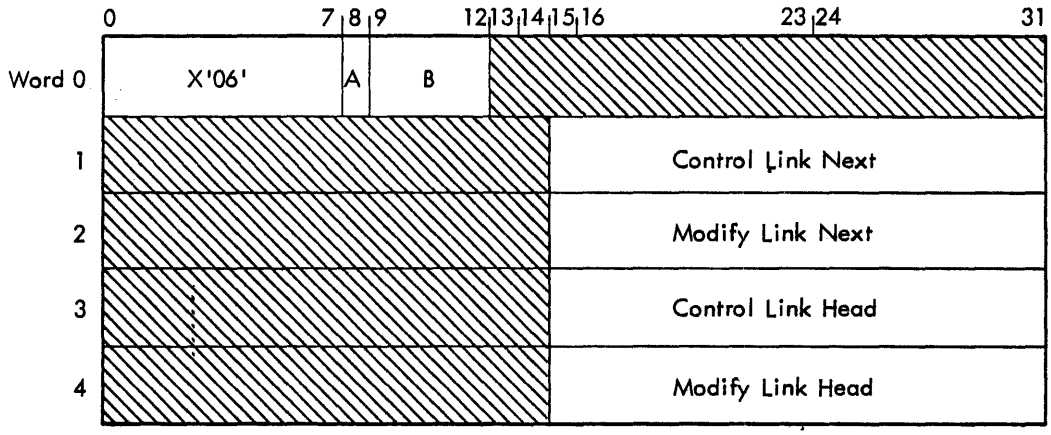
D = 1 if there is a check on range.

E = 1 if there is a check on picture.

F = 1 if this is an inverted item (bit C is set) and the secondary index group has been omitted (i.e., item cannot be modified).

G = 1 if item is a sort key in a set which is omitted (i.e., item cannot be modified).

Figure B-6. Item Definition

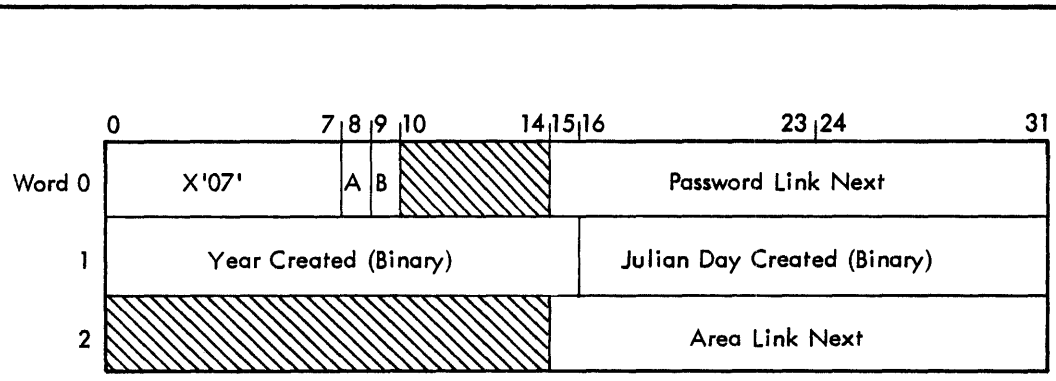


where

A is match indicator: 0 if equal; 1 if range.

B is control type: 0010 implies calc; 0100 – ascending sort; 0110 – descending sort.

Figure B-7. Control Definition

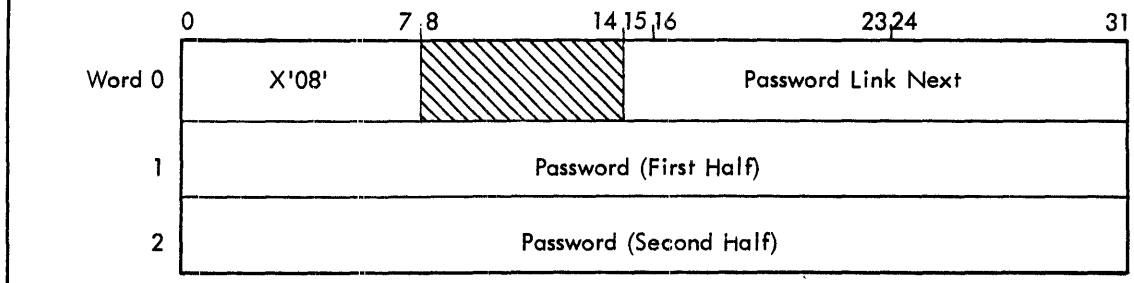


where

A = 1 if set pointers are four bytes long (i.e., multiple area database).

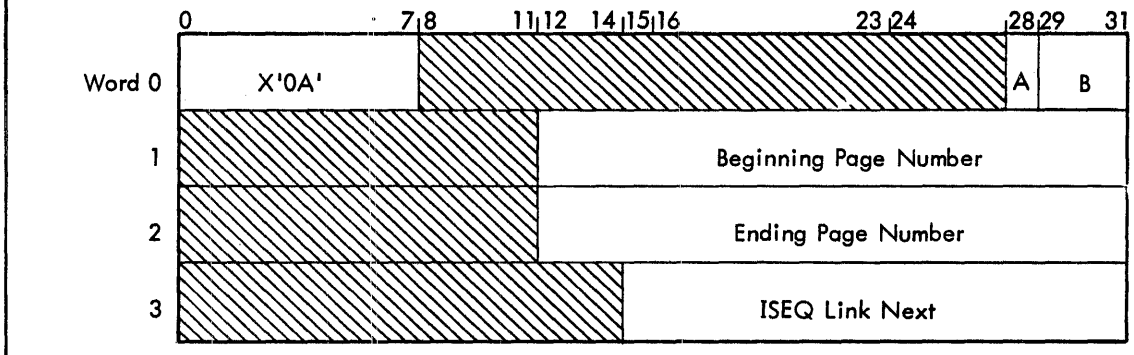
B = 1 if subschema was created using the COMPONENTS ARE ALL option on the subschema entry.

Figure B-8. Subschema Definition



Words 3 through 10 contain the retrieve authority indicators. Words 11 through 18 contain the update authority indicators.

Figure B-9. Password Definition

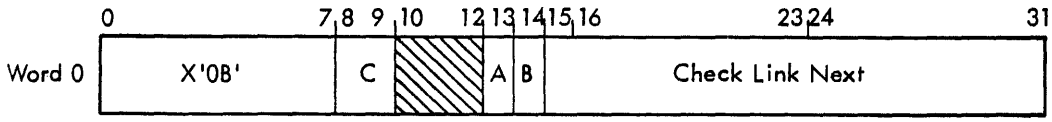


where

A = 0 if overflow range, = 1 if index level.

B = index level number, 0 through 8. (Note that index level zero actually contains the page range of the indexed data group.)

Figure B-10. Indexed-Sequential (ISEQ) Definition



Words 1 through N contain the check value(s), with N calculated as follows:

- If check value is a picture, N = 8. The picture is in TEXTC format starting at byte 0 of word 1.
- If check value is a range, N is based on the item type

Item Type	N
Binary	2
Floating short	2
Floating long	4
Packed	8
EBCDIC	8

If the item type is binary or floating short, the low/high values will be in words 1 and 2 and the total definition size will be three words.

If the item type is floating long, the low value will be in words 1 and 2 and the high value in words 3 and 4. Total definition size will be five words.

If the item type is packed decimal, the low/high range values will be in packed format and always 16 bytes in length. If item type is EBCDIC, the low/high range values will be left-justified in a 16-byte field and blank filled. In both of these cases, the low value will be in words 1 through 4, the high value in words 5 through 8, and total definition size will be nine words.

A = 1 if check value is PICTURE.

B = 1 if check value is RANGE.

C is definition size code: 0 implies three words; 1 – five words; 2 – nine words.

Figure B-11. Check Definition

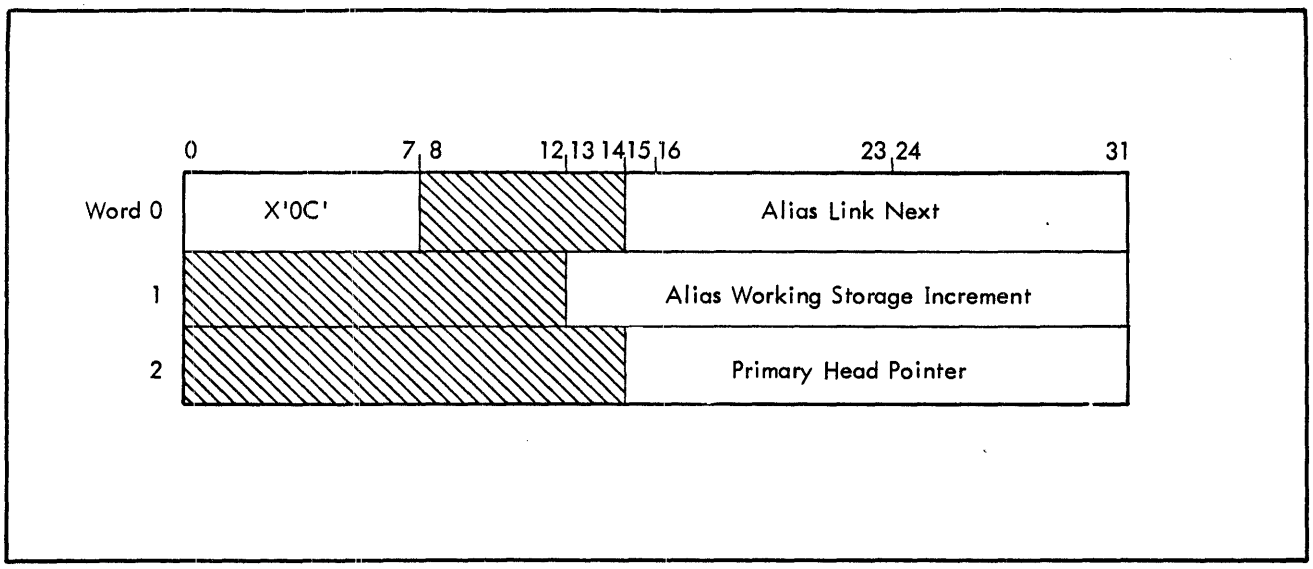
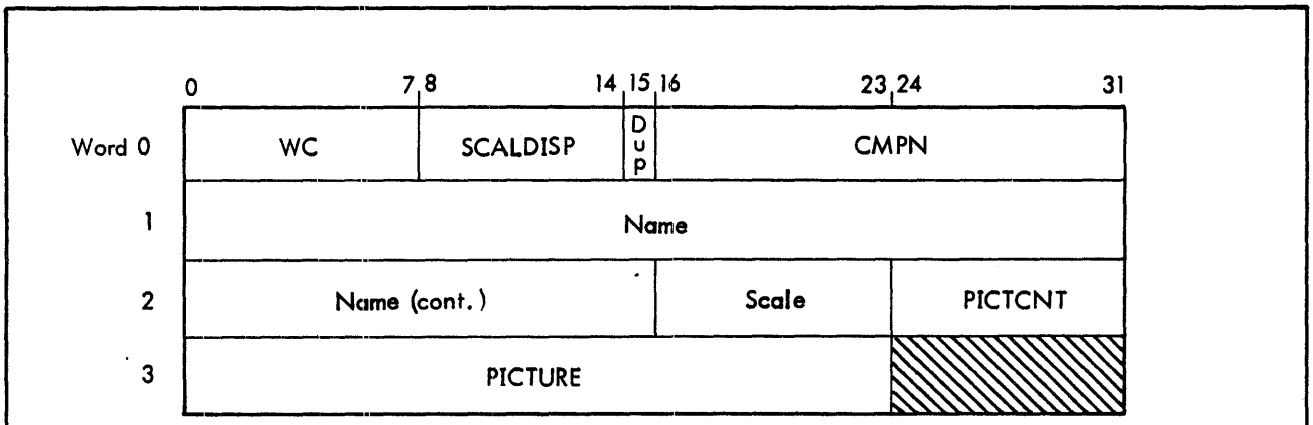


Figure B-12. Alias Definition



WC = Word count (zero indicates last entry).

SCALDISP = Byte displacement to scaling factor (zero implies no scaling factor and hence no PICTURE).

Dup = 1 if duplicates exist (i.e., name must be qualified).

CMPN = Subschema increment of component.

Name = DATA NAME with one trailing blank.

Scale = Number of fractional digits (negative value indicates unused integral digits.

PICTCNT = Length of PICTURE (bytes).

PICTURE = Character image of PICTURE.

Figure B-13. Name Table Entry Format

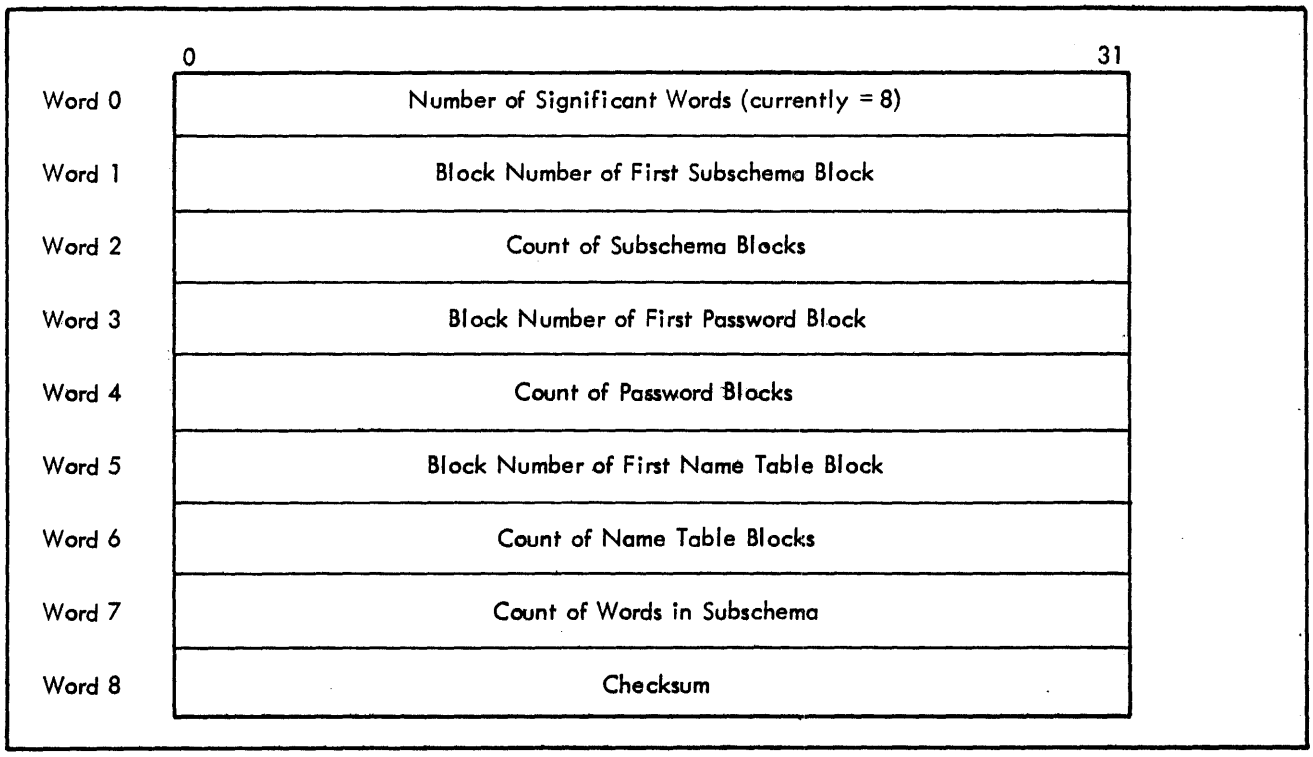


Figure B-14. Subschema File Directory Block Format (Blockzero)

APPENDIX C. SAMPLE DATABASE DEFINITION

This appendix illustrates (in Figures C-1 through C-7) the various aspects of the database definition function, and the operation of DMSFDP. The schema DDL for the sample database pictured in Figure 1 in the text is included, and two subschema DDL configurations using the schema are shown. The DMSFDP outputs in each of its two phases are shown, including a COPY and a SYSTEM listing.

```
1***200*** EXTENDED DMS FILE DEFINITION PROCESSOR -- VERSION A00.
 .
 ***210*** EXTENDED DMS SCHEMA DDL.
    1:  /*  THE DDL CONTAINED IN THIS FILE IS ERROR FREE. */
    2:
    3:  SCHEMA NAME IS SAMPLESCHEMA; PRIVACY LOCK FOR
    4:       EXTRACT IS 'EXLOCK'; PASSWORD IS 'PASWRD1'
    5:       RETRIEVE KEYS ARE 1,17,25 UPDATE KEY IS 231,247
    6:       PASSWORD IS 'PASWRD2' RETRIEVE KEY IS 3
    7:       UPDATE KEYS ARE 56,97,76.
    8:
    9:  AREA NAME IS AREA-1 CONTAINS 100 PAGES; NUMBER IS 1
   10:       ; INVENTORY 75
   11:       ; CHECKSUM IS NOT REQUIRED; JOURNAL IS NOT
   12:       REQUIRED; ENCIPHERING IS NOT REQUIRED.
   13:
   14:  AREA NAME IS AREA-2 CONTAINS 50 PAGES; NUMBER
   15:       IS 2; INVENTORY
   16:       PERCENT IS 50; JOURNAL IS NOT REQUIRED.
   17:
   18:  GROUP NAME IS GROUP-1 WITHIN AREA-1; NUMBER IS
   19:       100; LOCATION MODE IS DIRECT; PRIVACY LOCK
   20:       FOR RETRIEVE IS 1; PRIVACY LOCK FOR UPDATE IS
   21:       231; STATISTICS ARE REQUIRED.
   22:
   23:  GROUP NAME IS GROUP-2 WITHIN AREA-1 RANGE IS 1 THRU
   24:       30; NUMBER IS 200; LOCATION MODE IS VIA
   25:       SET-A SET PRIVACY LOCK FOR RETRIEVE IS 17
   26:       PRIVACY LOCK FOR UPDATE IS 231.
   27:  ITEM-21; PICTURE IS A(16); TYPE IS CHARACTER
   28:       PRIVACY LOCK FOR RETRIEVE IS 17; PRIVACY
   29:       LOCK FOR UPDATE IS 231.
   30:       ITEM-22 TYPE IS BINARY; OCCURS 4.
   31:  ITEM-23; TYPE IS FLOATING LONG.
   32:
   33:  GROUP NAME IS GROUP-3 WITHIN AREA-2 LOCATION MODE
   34:       IS CALC USING ITEM-32  DUPLICATES ARE ALLOWED;
   35:       NUMBER IS 300; PRIVACY LOCK FOR RETRIEVE
   36:       IS 17 PRIVACY LOCK FOR UPDATE IS 247.
   37:  ITEM-31 PICTURE IS X(31) OCCURS 4 TIMES.
   38:  ITEM-32; TYPE IS CHARACTER,31.
   39:
   40:  GROUP NAME IS GROUP-4 WITHIN AREA-2 RANGE IS 1 THRU 25
   41:       NUMBER IS 400; LOCATION MODE IS CALC USING
   42:       ITEM-41 DUPLICATES NOT ALLOWED.
   43:  ITEM-41 PICTURE IS 99V99;  PRIVACY LOCK FOR RETRIEVE
   44:       IS 1.
   45:  ITEM-42 PICTURE IS AA9(4)A.
   46:  ITEM-43 TYPE IS CHARACTER,4.
   47:  ITEM-44 TYPE IS BINARY.
```

Figure C-1. Schema DDL Listing for Sample Database

```
48!
49!  SET NAME IS SET-A; OWNER IS GROUP-1; ORDER IS FIRST.
50!  MEMBER IS GROUP-2
51!       ;INCLUSION IS AUTOMATIC SET OCCURRENCE SELECTION
52!       IS LOCATION MODE OF OWNER.
53!
54!  SET NAME IS SET-B; OWNER IS GROUP-2
55!       ORDER IS NEXT; STATISTICS ARE REQUIRED.
56!  MEMBER IS GROUP-3 INCLUSION IS AUTOMATIC
57!       ;LINKED TO OWNER; SET OCCURRENCE SELECTION
58!       IS THRU CURRENT OF SET.
59!
60!  SET NAME IS SET-C; ORDER IS NEXT
61!       ;OWNER IS GROUP-2; LINKED TO PRIOR
62!       ;STATISTICS ARE REQUIRED.
63!  MEMBER IS GROUP-4 INCLUSION IS MANUAL
64!       SELECTION IS THRU CURRENT OF SET.
65!
66!  SET NAME IS SET-D;OWNER IS GROUP-3
67!       ; ORDER IS SORTED; STATISTICS ARE
68!       REQUIRED.
69!  MEMBER IS GROUP-4 INCLUSION IS AUTOMATIC
70!       ;LINKED TO OWNER
71!       ;SET OCCURRENCE SELECTION IS THRU LOCATION
72!       MODE OF OWNER; ASCENDING RANGE KEY IS
73!       ITEM-41 DUPLICATES ARE NOT ALLOWED.
74!
75!  END.
```

Figure C-1. Schema DDL Listing for Sample Database (cont.)

```
***207*** SCHEMA CONTAINS 0005 PAGES.
***208*** THERE WERE 0000 DIAGNOSTIC MESSAGES.
                                STORAGE REQUIREMENT SUMMARY
                AREA NUMBER     DATA PAGES      INDEX PAGES     INVENTORY PAGES
                    01           0000100         0000000         0000001
                    02           0000050         0000000         0000001
***201*** SCHEMA GENERATION COMPLETE.
```

Figure C-2. Schema Generation Summary Output for Sample Database

```
1***200*** EXTENDED DMS FILE DEFINITION PROCESSOR -- VERSION A00.
 .
 ***202*** EXTENDED DMS SUBSCHEMA DDL.
    1!  SUBSCHEMA NAME IS COBOLSUB OF SCHEMA SAMPLESCHEMA
    4!       COMPONENTS ARE SPECIFIED.
    5!
    6!  SETS ARE SET-A SET-B SET-C.
    7!
    8!  AREAS ARE ALL; COMPONENTS ARE SPECIFIED.
    9!
   10!  GROUP NAME IS GROUP-1; COMPONENTS ARE ALL.
   11!
   12!  GROUP NAME IS GROUP-2-R RENAMES GROUP-2; COMPONENTS ARE SPECIFIED.
   13!       03  ITEM-21-22-23.
   14!           05  ITEM-21C, RENAMES ITEM-21.
   15!           05  ITEM-22-23.
   16!               07  ITEM-22-ALT.
   17!                   11  ITEM-22.
   18!               07  ITEM-23.
   19!
   20!  GROUP NAME IS GROUP-3; COMPONENTS ARE SPECIFIED.
   21!
   22!  GROUP IS GROUP-4; COMPONENTS ARE SPECIFIED.
   23!       02  ITEM-41.
   24!
   25!  END.
 ***214*** SUBSCHEMA FILE OCCUPIES 003 GRANULES.
 ***215*** IN CORE SUBSCHEMA REQUIRES 001 CORE PAGES.
 ***208*** THERE WERE 0000 DIAGNOSTIC MESSAGES.
 ***203*** SUBSCHEMA GENERATION COMPLETE.
```

Figure C-3. Subschema-1 DDL and Summary Output for Sample Database

```
    01  CCB.
        02  REF-CODE COMP VALUE ZERO.
        02  PAGE-NO  PIC 9(8).
        02  LINE-NO  PIC 9(3).
        02  FRST-REF COMP.
        02  LAST-REF COMP.
        02  GRP-NO   COMP.
        02  ERR-CODE COMP.
        02  ERR-NO   COMP.
        02  ERR-REF  COMP.
        02  PASSWORD PIC X(8) VALUE SPACES.
        02  AREA-NO  PIC 99.
    01  SET-TABLES COMP.
        02  SET-A.
            03  SET-OWNR.
            03  SET-PRIR.
            03  SET-CURR.
            03  SET-NEXT.
            03  SET-GRP.
        02  SET-B.
            03  SET-OWNR.
            03  SET-PRIR.
            03  SET-CURR.
            03  SET-NEXT.
            03  SET-GRP.
        02  SET-C.
            03  SET-OWNR.
            03  SET-PRIR.
            03  SET-CURR.
            03  SET-NEXT.
            03  SET-GRP.
    01  AREA-TABLE.
        02  AREA-1 PIC X(4) VALUE SPACES.
        02  AREA-2 PIC X(4) VALUE SPACES.
    01  GROUP-1.
        02  CURR-100 COMP.
    01  GROUP-2-R.
        03  ITEM-21-22-23.
```

Figure C-4. COPY Listing Corresponding to Subschema-1 for Sample Database

```
           05  ITEM-21C PIC A(16).
           05  ITEM-22-23.
               07  ITEM-22-ALT.
               11  ITEM-22 COMP OCCURS C04 TIMES.
               07  ITEM-23 COMP-2.0075).
       03  CURR-2CO COMP.
   01  GROUP-3.
       03  CURR-3CO COMP.
   01  GROUP-4.
       02  ITEM-41 PIC 99V99.
       02  CURR-4CC COMP.
   01  AREA-MASTERS-02 COMP.
       02  CURR-1COO.
       02  CALCSET.
           03  SET-OWNR.
           03  SET-PRIR.
           03  SET-CURR.
           03  SET-NEXT.
           03  SET-GRP.
   01  STATISTICS COMP.
       02  GRP-STATS-100 COMP.
           03  STAT-CTRL.
           03  STAT-ACC.
           03  STAT-INS.
           03  STAT-DEL.
       02  SET-STATS-C002 COMP.
           03  STAT-CTRL.
           03  STAT-NEXT.
           03  STAT-PRIR.
           03  STAT-HEAD.
       02  SET-STATS-C003 COMP.
           03  STAT-CTRL.
           03  STAT-NEXT.
           03  STAT-PRIR.
           03  STAT-HEAD.
```

Figure C-4. COPY Listing Corresponding to Subschema-1 for Sample Database (cont.)

```
1***2C0*** EXTENDED DMS FILE DEFINITION PROCESSOR -- VERSION A00.
 .
 ***2C2*** EXTENDED DMS SUBSCHEMA DDL.
    1!  SUBSCHEMA NAME IS METASUB OF SCHEMA SAMPLESCHEMA
    4!        ; COMPONENTS ARE SPECIFIED.
    5!
    6!  SET IS SET-0.
    7!
    8!  AREA IS AREA-2 COMPONENTS ARE SPECIFIED.
    9!
   10!  GROUP NAME IS FIRST, RENAMES GROUP-3; COMPONENTS ARE ALL.
   11!
   12!  GROUP NAME IS SECOND, RENAMES GROUP-4; COMPONENTS ARE
   14!        ITEM-41.
   15!        ITEM-44.
   16!
   17!  END.
 ***214*** SUBSCHEMA FILE OCCUPIES 003 GRANULES.
 ***215*** IN CORE SUBSCHEMA REQUIRES 0C1 CORE PAGES.
 ***2C8*** THERE WERE COCO DIAGNOSTIC MESSAGES.
 ***2C3*** SUBSCHEMA GENERATION COMPLETE.
```

Figure C-5. Subschema-2 DDL and Summary Output for Sample Database

```
          BOUND 8
CCB       RES       0
REF$CHDF&CCB DATA  0
PAGE$NB&CCB  RES,1 8
LINE$NB&CCB  RES,1 4
FRST$REF&CCB RES,1 4
LAST$REF&CCB RES,1 4
GRP$NB&CCB   RES,1 4
ERR$CHDF&CCB RES,1 4
ERR$NB&CCB   RES,1 4
ERR$REF&CCB  RES,1 4
PASSWORD&CCB DATA  '
AREA$NB&CCB  RES,1 4
          BOUND 8
SET$TABLES RES 0
SET$OWNR EQU 0
SET$PRIR EQU 1
SET$CURR EQU 2
SET$NEXT EQU 3
SET$GRP  EQU 4
SET$D DATA 0,0,0,0,0
          BOUND 8
AREA$TABLE RES 0
AREA$2 DATA '    '
          BOUND 8
FIRST RES 0
ITEM$31 RES,1 0124
ITEM$32 RES,1 0031
          BOUND 4
CURR$3CO RES,1 4
          BOUND 8
SECND RES 0
ITEM$41 RES,1 00C4
          BOUND 4
ITEM$44 RES,1 00C4
          BOUND 4
CURR$4CO RES,1 4
          BOUND 8
AREA$MASTERS$02 RES 0
          BOUND 4
CURR$1CCC&02 RES,1 4
CALCSET&02 DATA 0,0,0,0,0
          BOUND 8
STATISTICS RES 0
STAT$CTRL EQU 0
STAT$ACC,STAT$NEXT EQU 1
STAT$INS,STAT$PRIR EQU 2
STAT$DEL,STAT$HEAD EQU 3
SET$STATS$00C4 DATA 0,0,0,0
          END
```

Figure C-6. SYSTEM Corresponding to Subschema-2 for Sample Database

APPENDIX D. DATABASE PAGE FORMATS

This appendix contains Figures D-1 through D-5, showing details of the various page formats in an EDMS database.

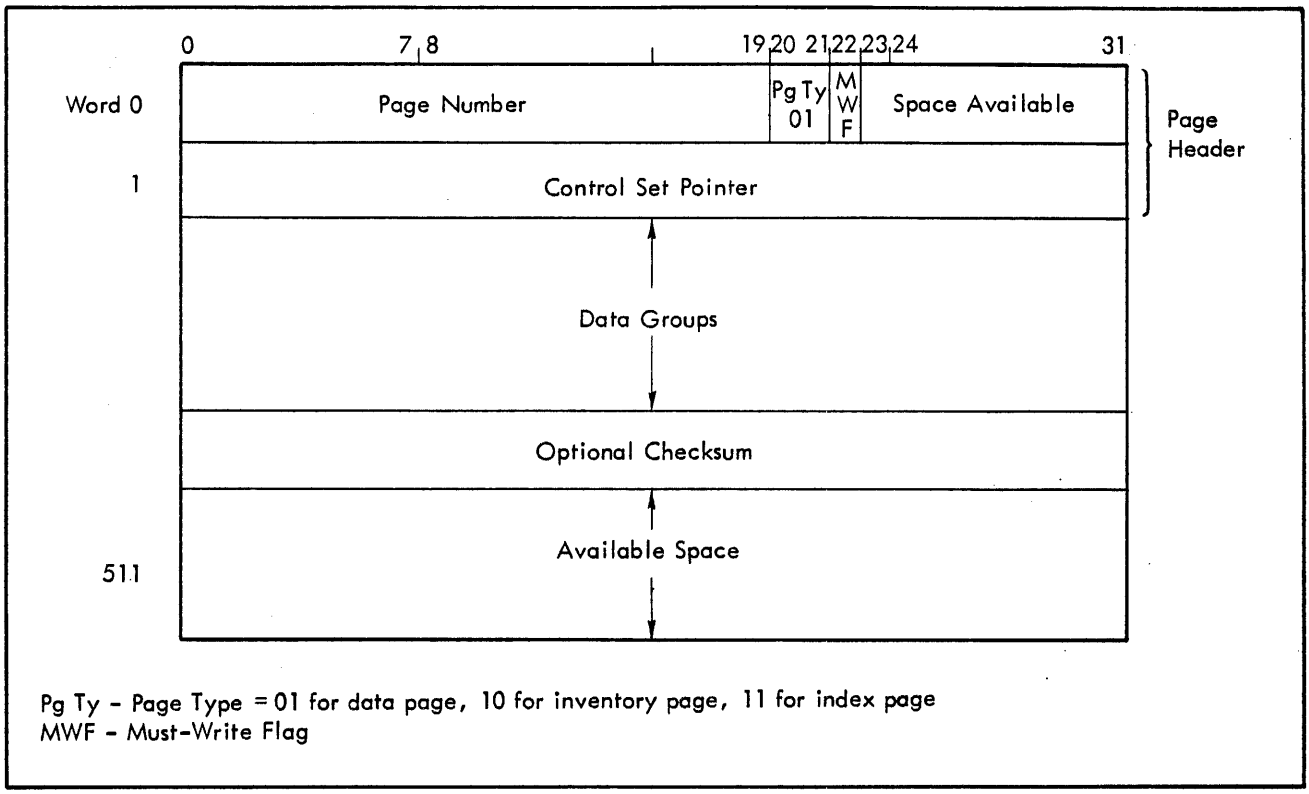


Figure D-1. Data Page Format

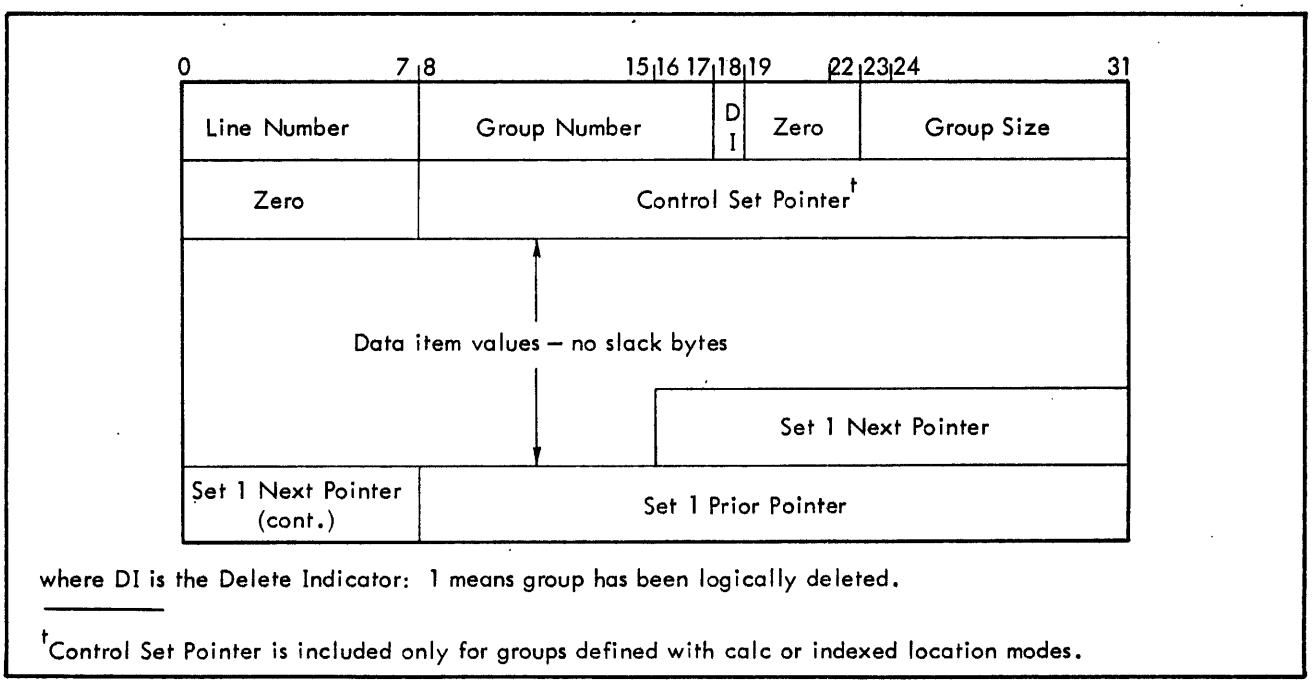
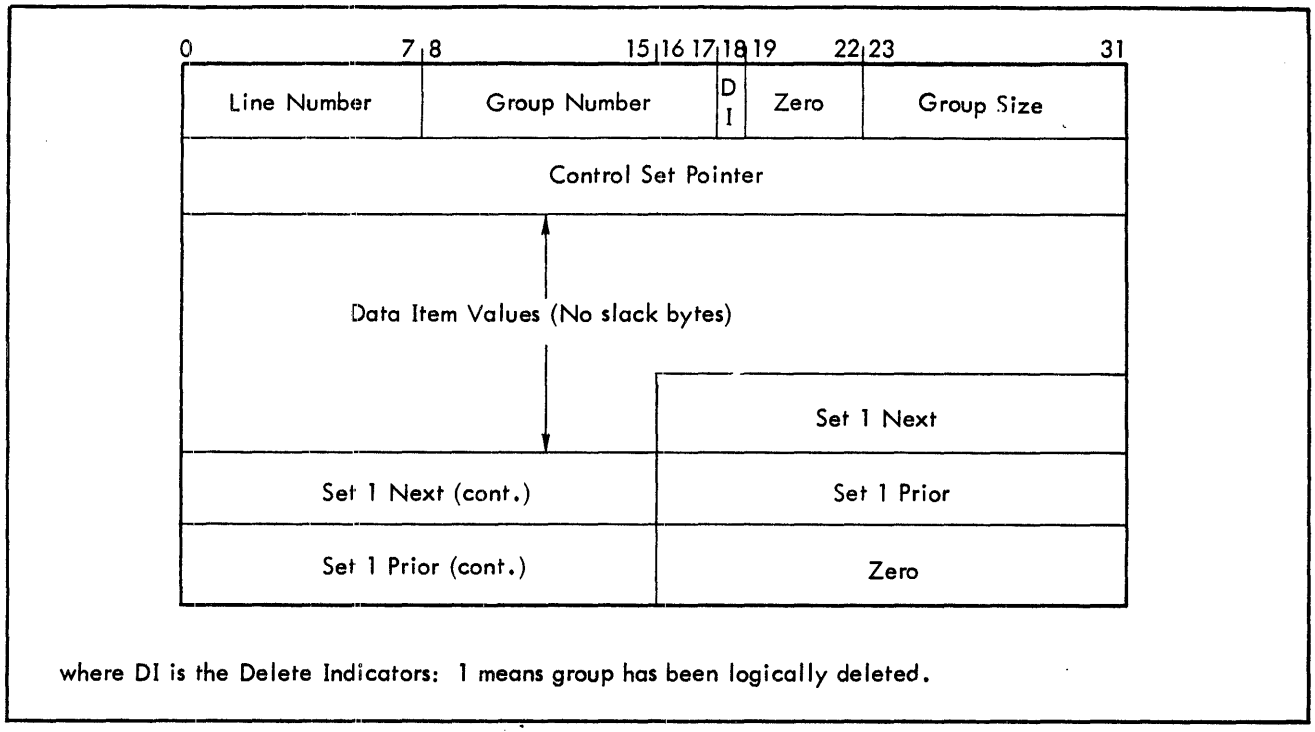
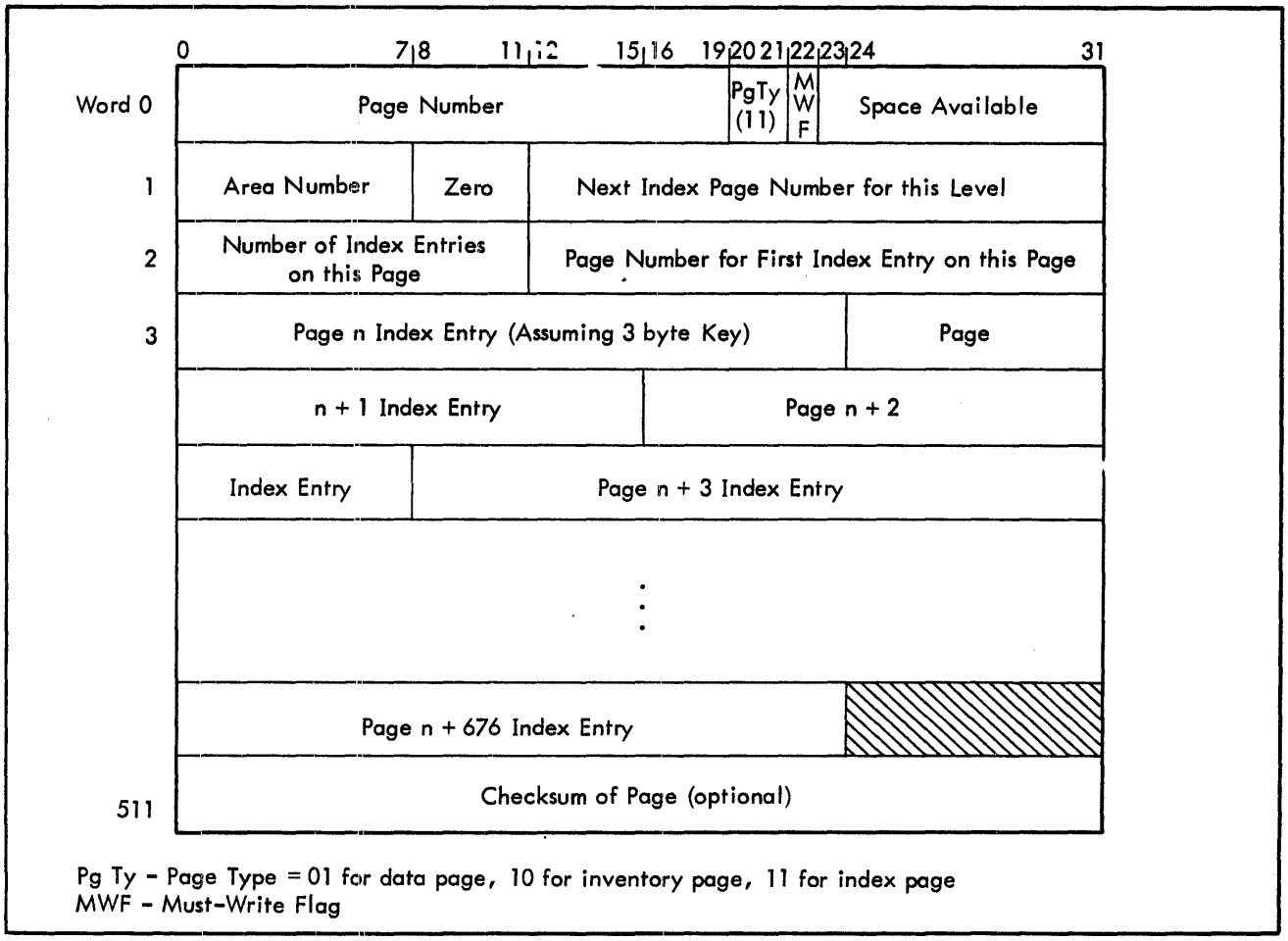


Figure D-2. Data Group Occurrence with Three-Byte Set Pointers



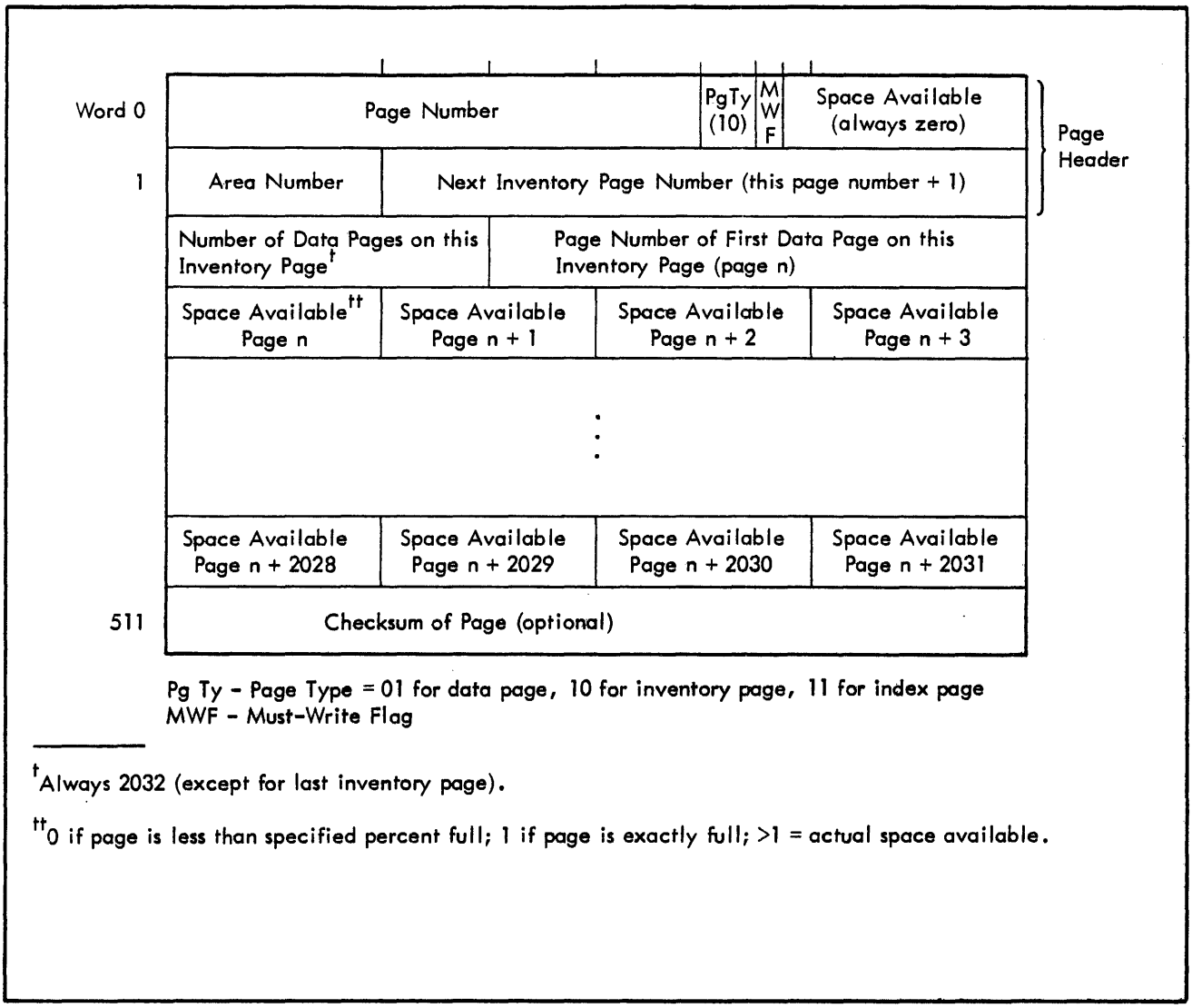
where DI is the Delete Indicators: 1 means group has been logically deleted.

Figure D-3. Data Group Occurrence with Four-Byte Set Pointers



Pg Ty - Page Type = 01 for data page, 10 for inventory page, 11 for index page
MWF - Must-Write Flag

Figure D-4. Index Page Format



Pg Ty - Page Type = 01 for data page, 10 for inventory page, 11 for index page
MWF - Must-Write Flag

†Always 2032 (except for last inventory page).

††0 if page is less than specified percent full; 1 if page is exactly full; >1 = actual space available.

Figure D-5. Inventory Page Format

APPENDIX E. SEQUENTIAL FILE FORMATS

This Appendix describes the two types of sequential files that are generated and processed by Extended DMS, the Journal/Dump file and the Statistics file.

Sequential files of the Journal/Dump format are created by the DBM during user program (journal) and by the dump and load utilities (dump format). Journal/Dump files have records in three formats: Begin records, End records, and Page-Image records. Figures E-1 through E-3 illustrate these individual records. Figure E-4 shows a summary of the three.

Statistics files are created by the DBM during program operation. and contain records in four formats: Job ID records, Area records, Group records, and Set records. These records are illustrated in Figures E-5 through E-8.

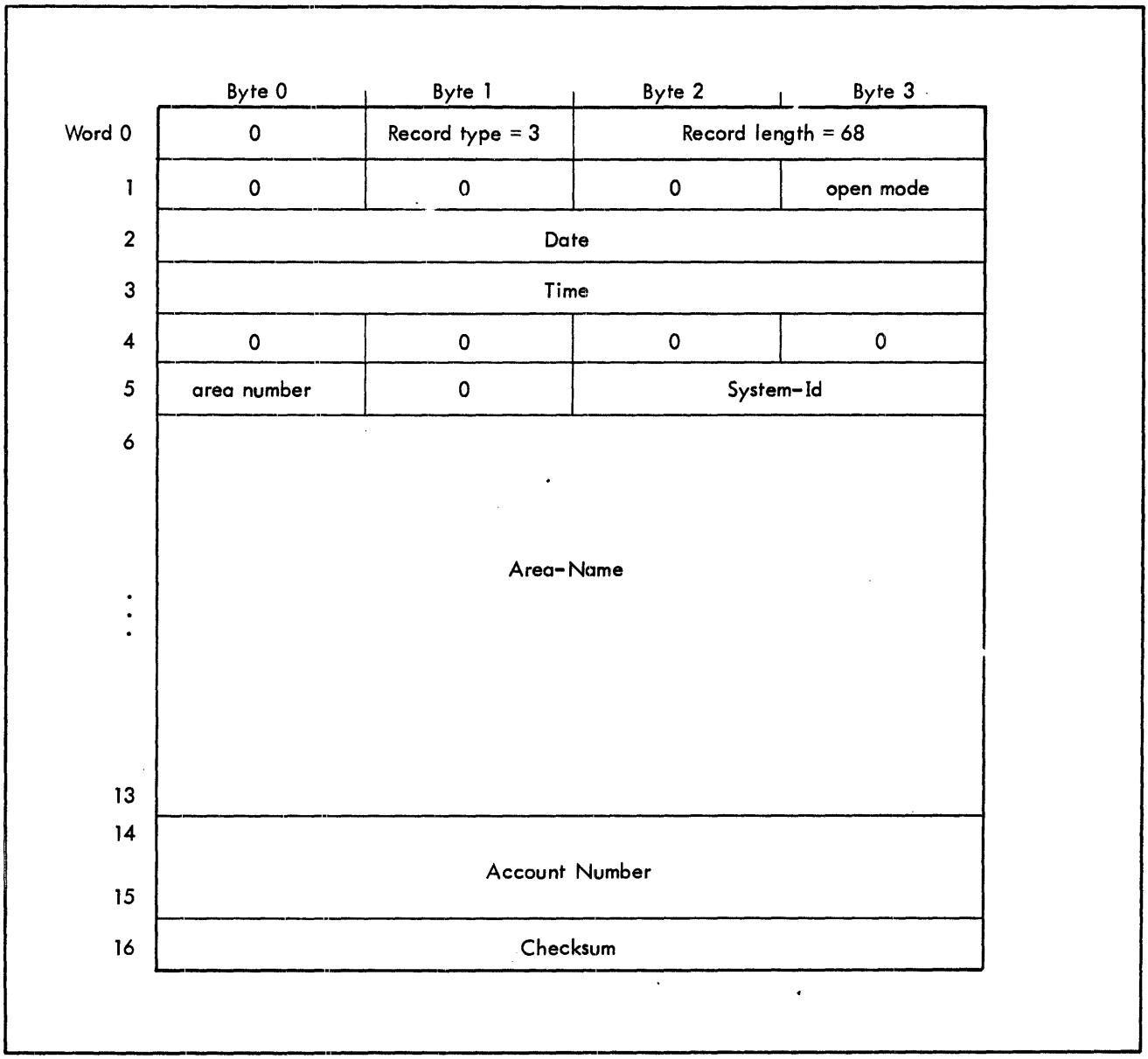


Figure E-1. Journal/Dump Begin Record

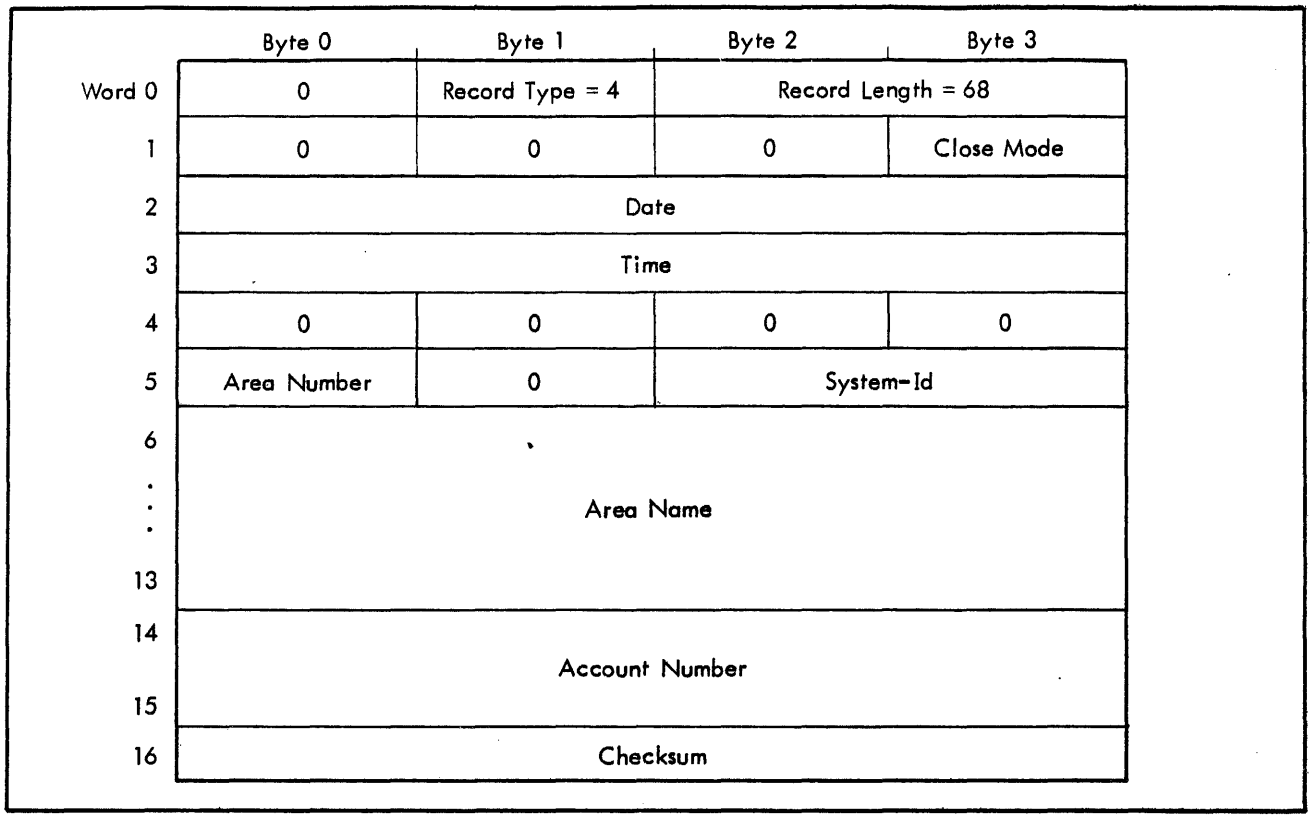
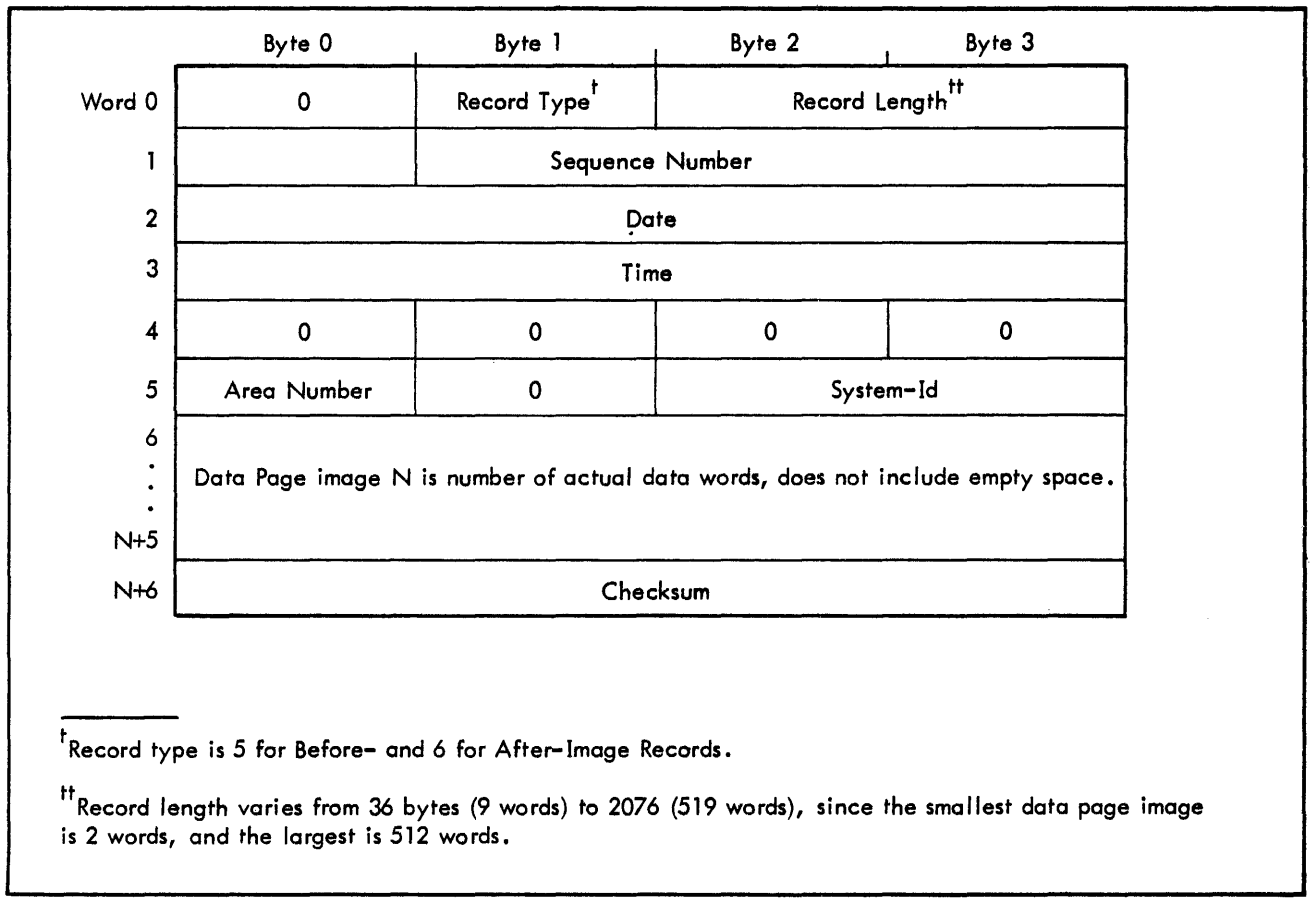


Figure E-2. Journal/Dump End Record



†Record type is 5 for Before- and 6 for After-Image Records.

††Record length varies from 36 bytes (9 words) to 2076 (519 words), since the smallest data page image is 2 words, and the largest is 512 words.

Figure E-3. Journal/Dump Page-Image Record

Word	Byte	Begin	End	Before or After
0	0	MBZ[†]	MBZ[†]	MBZ[†]
	1	Record type (3)	Record type (4)	Record type (=5 before; =6 after)
	2-3	Record length in bytes	Record length in bytes	Record length in bytes
1	0-2	MBZ[†]	MBZ[†]	Sequence number
	3	Open mode	Close mode	
2	0-3	Date	Date	Date
3	0-3	Time	Time	Time
4	0-3	MBZ[†]	MBZ[†]	MBZ[†]
5	0	Area number	Area number	Area number
	1	MBZ[†]	MBZ[†]	MBZ[†]
	2-3	System-Id	System-Id	System-Id
6-13		Area-name	Area-name	tt
14-15		Account number	Account number	
16		Checksum	Checksum	

where

Record length (word 0) is that of journal record. (Record size varies from 9 to 519 words).

Open mode (word 1) = 1 for retrieve, 2 for update, 3 for create, 4 for DMSDUMP.

Close mode (word 1) = 0 for normal, 1 for abnormal.

Sequence number (word 1) -- before start, at -1 and decrementing; after start, at +1 and incrementing.

Date (word 2) is binary halfword year and binary halfword Julian day.

Time (word 3) is binary value HHMMTTTT (hour, minute, calculated time approximately milliseconds since last minute).

Word 4 is reserved for use in future enhancements.

System-Id (word 5) is a two-byte binary value.

[†]Must be zero.

[tt]Each Before/After record contains a data page image in words 6 through N+5, and a checksum in word N+6 (where N is the number of data words actually stored on the page).

Figure E-4. Journal/Dump File Format Summary

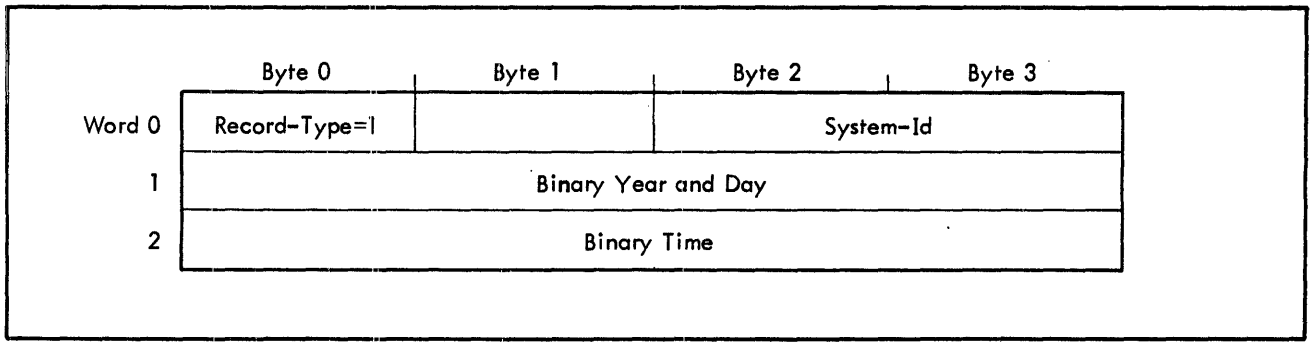


Figure E-5. Statistics Job Id Record

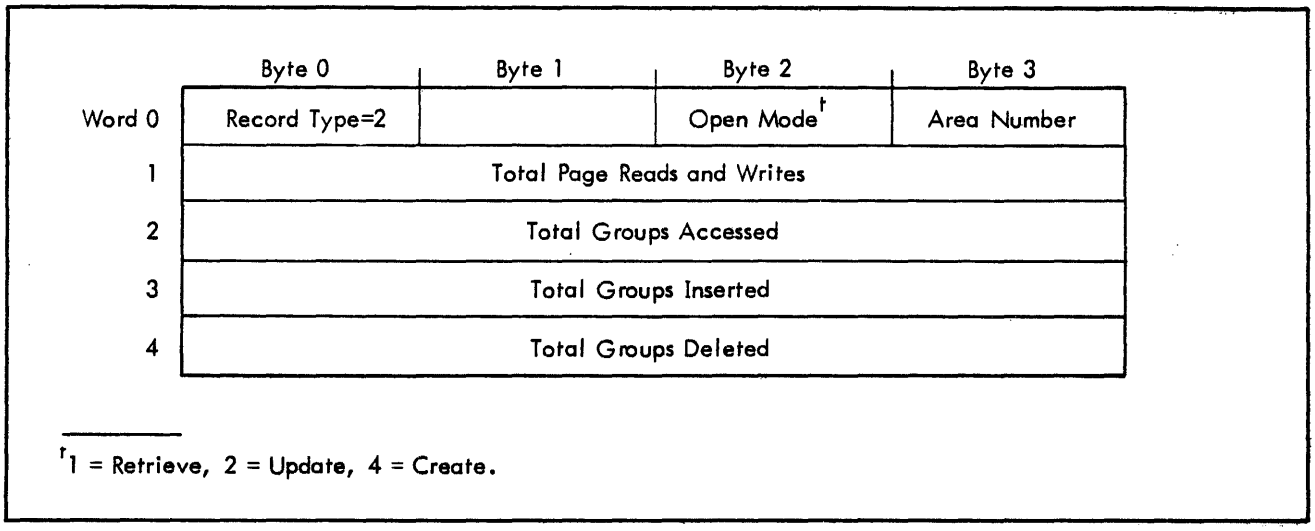


Figure E-6. Area Statistics Record

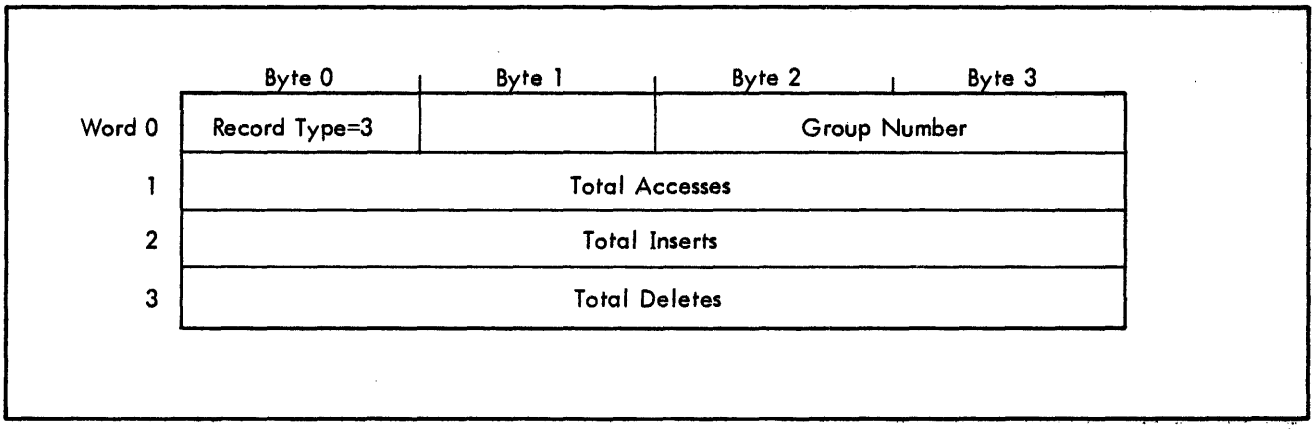


Figure E-7. Group Statistics Record

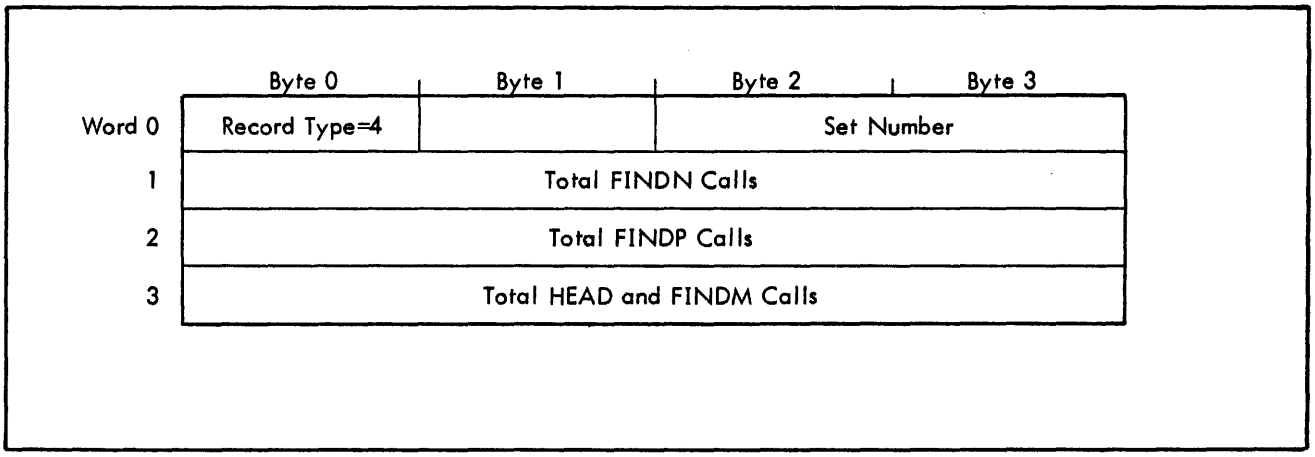


Figure E-8. Set Statistics Records

APPENDIX F. ERROR MESSAGES

This appendix contains error messages generated by the EDMS File Definition Processor, the Database Manager, and the EDMS utility routines, as follows:

Source	Table
DMSFDP	F-1
DBM, Data-Dependent	F-2
DBM, Non-Data-Dependent	F-3
DMSINIT	F-4
DMSDUMP	F-5
DMSLOAD	F-6
DMSSUMS	F-7
RPCL	F-8
DMSREST	F-9
APL/EDMS	F-10

Table F-1. DMSFDP Error Messages

Message	Meaning
100 REDUNDANT CLAUSE NOT ALLOWED.	A clause other than password, check, ascending/descending, or condition was repeated in a subentry.
101 WITHIN CLAUSE MISSING.	A schema-DDL group or invert subentry did not specify the area that is to contain occurrences.
102 NUMBER CLAUSE IS MISSING.	An area, group, or invert subentry did not specify a unique identifier for the area or group.
103 LOCATION CLAUSE IS MISSING.	A group subentry did not specify a location mode (direct, calc, indexed, or via) for the group.
105 OWNER CLAUSE IS MISSING.	A set subentry did not identify a group to participate as owner.
106 ORDER CLAUSE IS MISSING.	A set subentry did not specify logical sequence (first, last, next, or sorted) for set occurrences.
107 INCLUSION CLAUSE IS MISSING.	A member subentry did not specify whether inclusion of member occurrences in set occurrences would be automatic or manual.
108 SELECTION CLAUSE IS MISSING.	A member subentry did not specify the method (current or location mode of owner) of identifying set occurrences for linking member occurrences.

Table F-1. DMSFDP Error Messages (cont.)

Message	Meaning
109 DUPLICATES CLAUSE/SUBCLAUSE MISSING.	Clause was not included in an invert subentry; or subclause was not included with calc location mode in a group subentry, or with ascending/descending sort keys for a member of a sorted key.
110 USING SUBCLAUSE MISSING.	Calc or indexed location mode in group subentry did not name control items.
111 COMPONENTS CLAUSE IS MISSING.	A subschema-DDL subschema, area, or group entry did not indicate if components were all or specified.
204 REDUNDANT OPTION -- ILLEGAL.	A control option was repeated.
205 ILLEGAL OPTION.	A control card option was not a DMSFDP option.
206 NOSCHEM OPTION IGNORED -- NO SCHEMA DDL. ***218*** NOSUB OPTION IGNORED -- NO SUBSCHEMA DDL. ***219*** NOCBL OPTION IGNORED -- NO SUBSCHEMA DDL OR NO COPY FILE NAME. ***220*** NOMETA OPTION IGNORED -- NO SUBSCHEMA DDL OR NO SYSTEM FILE NAME. ***221*** NOLIST OPTION IGNORED -- NO SUBSCHEMA DDL OR NO SYSTEM OR COPY FILE NAMES.	A control card option has specified suppression of an output that could not have resulted from the inputs in any case.
301 SYNTAX ERROR.	Any of several errors, such as illegal characters, misspelling, use of a reserved word as a name, etc.
302 AREA ENTRY OUT OF ORDER. ***303*** GROUP ENTRY OUT OF ORDER. ***304*** ITEM ENTRY OUT OF ORDER. ***305*** INVERT ENTRY OUT OF ORDER. ***306*** SET ENTRY OUT OF ORDER. ***307*** MEMBER SUBENTRY OUT OF ORDER. ***308*** END ENTRY OUT OF ORDER.	The DDL-required entry/subentry order has been violated. This may have resulted from an entry being discarded for errors.
309 ONLY ONE SCHEMA/SUBSCHEMA ALLOWED.	More than one was included.
310 UNEXPECTED END OF FILE. PROCESSING TERMINATED.	The last entry processed was not an end entry.
311 PRECEDING ENTRY HAS BEEN DISCARDED BECAUSE OF ERRORS.	This may cause succeeding entries to be out of order.

Table F-1. DMSFDP Error Messages (cont.)

Message	Meaning
401 SYMBOL TOO LONG.	A name was more than 30 characters long.
402 ILLEGAL VALUE.	An integer value, group number, area number, etc. was greater than the specified limits.
404 NON-UNIQUE AREA NAME.	The name specified in a schema-DDL area entry duplicated that of another area in the database.
405 NON-UNIQUE GROUP OR SET NAME.	The name specified in a schema-DDL group or set subentry duplicated the name of a previously defined group, set, or item.
406 UNDEFINED AREA.	The area named in a group or invert subentry within clause was not defined in an area entry.
407 TOO MANY CONTROL/SORT KEYS.	More than seven keys were specified in a calc or indexed location mode specification, or in ascending/descending clauses in a member subentry.
408 CONTROL ITEM item-name FOR group-name GROUP IS UNDEFINED.	The item identified by item-name was designated as a control item for location mode of calc or indexed, but was not defined in an item subentry for the group identified by group-name.
409 GROUP group-name INTERSECTS INDEX/ OVERFLOW RANGE.	The page range specified for the named group overlaps the range of an indexed group or the overflow range for the area.
410 MULTIPLE INDEXED GROUPS DEFINED IN THE SAME AREA.	Two or more subschema-DDL group subentries specified location mode of indexed and the same area-name in the within clauses.
411 NON-UNIQUE GROUP OR INVERT NUMBER.	The integer in a schema-DDL group or invert subentry number clause duplicated the number in a previous group or invert subentry.
412 UNDEFINED KEY.	A retrieve/update key in a schema-DDL group or item subentry did not match any key specified in a password clause.
413 ITEM NAME DUPLICATES GROUP OR SET NAME.	The name specified in a schema-DDL item subentry duplicated the name of a previously defined set or group.
414 ITEM NAME CANNOT BE UNIQUELY IDENTIFIED.	The name specified for an item results in a duplication even when qualified (two items within the same group with the same name).
415 PICTURE AND TYPE INCONSISTENT.	Specifications for picture and type in a schema-DDL item subentry conflicted (e.g., a numeric picture and character type).
416 ILLEGAL CHECK VALUE IN CHECK CLAUSE NUMBER nn.	A check clause in a schema-DDL item subentry contained an illegal value. The nn refers to the sequence of input of the clauses.

Table F-1. DMSFDP Error Messages (cont.)

Message	Meaning
417 group-name GROUP SIZE EXCEEDS ONE PAGE.	The combination of items (including occurs) defined for the named group resulted in a group size of more than 510 (or 509 if there is a checksum) words.
418 ITEM NOT DEFINED IN PRECEDING GROUP.	The item designated as secondary index in an invert subentry was not defined in an item subentry for the group.
419 MULTIPLE INVERT ENTRIES USE SAME ITEM.	The same item was specified as the secondary index item in two or more invert subentries.
420 UNDEFINED GROUP.	The group identified as owner in a set subentry, or as member in a member subentry was not defined in a group entry.
421 SIZE OF DATA ITEM INDETERMINATE.	A schema-DDL item subentry did not include a picture clause, and the type clause did not include or imply an item size.
422 TRUNCATION.	An integer value consisted of more than the legal number of digits (e.g., three digits used for area number).
423 MANUAL OR OPTIONAL INCLUSION ILLEGAL FOR SET WHICH GROUP IS VIA.	A schema-DDL member subentry specified manual or optional automatic inclusion and the group location mode is via the set.
424 UNDEFINED ITEM.	An item-name specified did not match any name specified in an item subentry.
425 SORT ITEM NOT DEFINED IN MEMBER GROUP.	An item designated as a set sort key in a member subentry was not defined in the group specified as member.
426 DATA ITEM NOT DEFINED IN OWNER GROUP.	The item for which an alias was specified in a member subentry was not defined in the group named in the owner subentry.
427 NON-UNIQUE ALIAS.	The same item-name was used for two or more alias clauses in a member subentry.
428 WARNING -- ALIASES FOR set-name SET INCONSISTENT WITH OWNER'S CONTROLS.	The aliases specified in a member subentry did not exactly correspond to the control items for the owner group. For example, the owner group was calc using four items, and only three were given aliases. This situation is not illegal, only dangerous, and does not interfere with schema generation.
429 FILL PCT/OVERFLOW RANGE USED IN AREA area-name WHICH HAS NO INDEXED GROUP.	No group defined as within the named area had a location mode of indexed, making the fill percent or overflow range specification meaningless.
430 STORAGE/VIA SET set-name UNDEFINED.	A schema-DDL group subentry specified the named set in a via location mode or in a storage subclause, but there was no set entry defining the set.

Table F-1. DMSFDP Error Messages (cont.)

Message	Meaning
431 STORAGE IS set-name SET FOR group-name -- AREA CONFLICT.	The owner group of the named set was defined as in a different area than the group identified by group-name, therefore the use of that set as the storage set is illegal.
432 ILLEGAL PICTURE.	The character-string in a picture clause was not a legal combination of characters.
433 ITEM SIZE EXCEEDS 255 BYTES PER OCCURRENCE.	The total size of the item in the DMS group occurrence would exceed maximum item-size.
434 CHECK ILLEGAL WITH OCCURS OR WHEN ITEM SIZE EXCEEDS 16 BYTES.	A restriction on the use of the check clause was violated.
435 CAN'T INVERT ON AN ITEM WHICH OCCURS.	An item that was defined with an occurs clause was specified as the secondary index item in an invert subentry.
436 ITEM WHICH OCCURS CAN'T BE CONTROL KEY.	An item defined with an occurs clause was designated as a set sort key.
437 GROUP group-name CONTROL ITEM item-name ILLEGAL OCCURS.	The named item, specified as the control item in a location mode using-subclause for the named group, was defined with an occurs clause.
438 MEMBER group-name IN SET set-name NEEDS SORT KEYS.	The named set was defined as sorted, but the member subentry designating the named group did not include ascending/descending clauses.
439 INCONSISTENT SORT KEY TYPE/SIZE FOR MEMBERS OF set-name SET.	The items specified as sort keys in member subentries for two or more groups did not correspond.
440 GROUP NUMBER USED AS SORT KEY ON set-name SET WHICH HAS BUT ONE MEMBER.	The order clause for the named set specified sorted with group number as major or minor, but only one group was identified as a member of the set.
441 OCCURRENCE SELECTION MUST BE CURRENT FOR AREA OWNER, OR FOR SETS ORDERED NEXT OR PRIOR.	The set occurrence selection clause in a member subentry violated one of the indicated restrictions.
442 MEMBER NOT IN AREA WHICH OWNS THIS SET.	Area-is-owner was specified for the set and a group designated as a member was not defined as within the area.
443 STORAGE IS set-name SET FOR group-name -- NOT MEMBER.	The named set was identified as the storage set in the group subentry defining the named group, but the group was not identified in a member subentry for the set.
444 GROUP CANNOT PARTICIPATE MORE THAN ONCE IN SINGLE SET.	A group was designated as both a member and an owner or as a member twice in the same set entry.
446 MEMBER group-name OF set-name SET GIVES ALIAS FOR item-name -- NOT CONTROL ITEM.	The member subentry for the named group included an alias subclause for an item that was not a control item for the owner of the named set.

Table F-1. DMSFDP Error Messages (cont.)

Message	Meaning
447 STORAGE MASTER FOR GROUP group-name NOT IN INDEXED DATA RANGE -- MUST BE.	The page range for the owner of the set specified in a storage subclause for the named group was not within the range specified for the area's indexed group.
448 NO STORAGE SET SUPPLIED FOR GROUP group-name.	The group subentry for the named group included a storage subclause, but the specified set was not defined in a set entry.
449 MEMBER group-name OF set-name SET NEEDS UNIQUE OWNER.	The member subentry for the named group in the named set specified location mode of owner for set occurrence selection, but the owner's location mode does not provide uniqueness.
450 ILLEGAL RANGE IN CHECK CLAUSE NUMBER nn. (I.E. LO > HI).	Range of values specified in improper order. The nn refers to sequence of input of the clauses.
451 MUST HAVE CHECKSUMS ON ENCIPHERED AREA.	Checksums were prohibited and enciphering requested in the same area entry.
452 SORT KEYS ARE NOT ALLOWED UNLESS SET ORDER IS SORTED.	A member subentry included ascending/descending keys but the set order specified in the set subentry was not sorted.
453 CHECK ON PICTURE ILLEGAL IF NO PICTURE CLAUSE.	An item subentry included a CHECK clause specifying PICTURE, but no PICTURE clause.
501 PRIVACY LOCK VIOLATION. PROCESSING TERMINATED.	An attempt was made to generate a subschema from an extract-protected schema without supplying the proper key in the subschema entry.
502 UNDEFINED OR DUPLICATE SET.	The subschema-DDL set entry named a set not defined by the schema or named the same set twice.
503 UNDEFINED OR DUPLICATE AREA.	An area not defined in the schema was specified in a subschema-DDL area entry or one area was named twice in one or more area entries.
504 GROUP IS IN AREA NOT DEFINED FOR SUBSCHEMA.	The group named in a subschema-DDL group entry was defined in the schema as within an area that is not defined in the subschema.
505 UNDEFINED OR DUPLICATE GROUP.	The group specified in a subschema-DDL group entry was not defined in the schema or the same group was named in two or more group entries.
506 'ALL' OPTION ILLEGAL HERE. SKIP TO NEXT'.'.	ALL was specified after specific areas were named in a subschema area entry.
508 GROUP IS IN AREA WHICH INCLUDES ALL COMPONENTS.	A subschema-DDL group entry specified a group that was defined as within an area for which a components clause indicated all.
509 SEC INDEX FOR item-name IN group-name IS IN OMITTED AREA.	The named item in the named group was designated a secondary index and the area that was to contain the invert group occurrences is not defined for the subschema.

Table F-1. DMSFDP Error Messages (cont.)

Message	Meaning
511 BAD SCHEMA -- GROUPRET EXISTS FOR ITEM NOT DEFINED IN GROUP. PROCESSING TERMINATED. ***512*** BAD SCHEMA -- CAN'T FIND SCHEMAHD. PROCESSING TERMINATED.	These two messages indicate a defective schema file. Neither should occur if the schema was generated correctly and not subsequently modified.
513 ILLEGAL LEVEL NUMBER.	The level number specified in a subschema-DDL item subentry did not conform to the rules for level-number sequence.
514 DUPLICATE ITEM.	The same item name was specified in two or more subschema-DDL item subentries.
515 SET set-name REQUIRES GROUP group-name.	The named set was selected for the subschema but the named group (which is the owner or a member of the set) was not.
516 LAST ITEM IN PRECEDING GROUP WAS NOT DEFINED.	The item name in a subschema-DDL item subentry did not refer to an item defined in the schema (not discovered until after processing had begun on the following group subentry).
517 RENAMES ILLEGAL WITH UNDEFINED ITEM NAME.	The item name in a renames clause was not the name of an item defined in the schema.
518 ILLEGAL ALPHANUMERIC LITERAL.	The size of the literal specified in a condition clause in a subschema-DDL item subentry was greater than the space allocated for it in a COPY record.
522 EXPECTED SUBSCHEMA ENTRY NOT FOUND. PROCESSING TERMINATED.	Input that followed the schema-DDL end was not a subschema entry.

Table F-2. DBM Data-Dependent Errors

Error Number	Error Condition
1	Space is insufficient to insert a new group occurrence in that portion of the data-base in which the group type may be placed.
2	An attempt was made by the DBM to retrieve an occurrence of a given group. The reference code used was from REF-CODE in the CCB, CURR-XXX for the group, or a set table for a set in which the group participates. The occurrence retrieved was not the group intended.
3	Attempt was made to retrieve a group on the basis of its location mode. The values supplied for the control items did not define a group occurrence.
4	Attempt was made to establish a group occurrence that violated a duplicate clause for the group.
5	Attempt to use FINDD with REF-CODE equal to zero.

Table F-2. DBM Data Dependent Errors (cont.)

Error Number	Error Condition
6	Reference code supplied for the FINDD call resulted in retrieval of a group occurrence that was logically deleted.
7	The reference code of a group occurrence to be retrieved is not present in the page.
8	Page number of a data page is outside the range of data pages for the area.
9	Attempt to retrieve a direct-group occurrence with value of REF-CODE equal to zero.
10	The area number portion of the reference code supplied for retrieval of a group occurrence is incorrect.
11	The area number portion of the reference code supplied for storing a direct-group occurrence is incorrect.
12	Attempt to traverse a set without establishing a position in the set because of the optional or manual status of the set member.
13	Attempt to use DELETSEL or REMOVSEL, with the object group occurrence the owner of a nonempty set occurrence.
14	Attempt to link a manual or optional group, with the object group occurrence already linked into an occurrence of the set.
15	Attempt to delink a manual or optional group, with the object group occurrence not linked into an occurrence of the set.
16	Attempt to store an indexed group in create mode, with the values for the index control items not greater than those already in the area.
17	Attempt to modify or store a data group where the values of a data item do not pass the data validation checks specified in the schema.
18	FINDDUP of a calc group resulted in inability to find a group having duplicate values for the calc control items.
19	An area was opened for retrieval and the database lockout bit was set.
20	Attempt to relink a manual or optional group, with the object group occurrence not linked into an occurrence of the set.

Table F-3. DBM Non-Data-Dependent Errors

Error Number	Error Condition
30	Monitor returned a deadlock indication on an attempt to enqueue at pages in a shared area.
31	Group to be retrieved or stored depends upon retrieval of a current owner group. The user has not retrieved an occurrence of the owner group.
32	Attempt to use a procedure whose object is the current of group type without having a current occurrence of the group type. The procedures are Get, Modify, Delete (all forms), Link, Delink, Relink, and FINDDUP.
33	Attempt was made to traverse a set with no current position in the set.

Table F-3. DBM Non-Data Dependent Errors (cont.)

Error Number	Error Condition
34	Attempt to use the HEAD procedure without a current position in the set.
35	Attempt to use the FINDC procedure without having a current of the group type.
36	Use of FINDG call with the object group occurrence having location mode via set. The set is not sorted.
37	Attempt to HEAD a set whose owner is defined to be the AREA.
38	Attempt to modify a data item that is an index control item.
39	Attempt was made to update an area of the database that was opened for retrieve only.
40	Procedural call to open any area while executing in another area.
41	Attempt to access an unopened area.
42	Procedural call without any areas open. The only calls allowed without an open area are DMSTRACE, ENDTRACE, DMSSTATS, ENDSTATS, RPTSTATS, DMSABORT, SETERR, RESETERR, and DMSLOCK.
43	Group referenced by FINDX or FINDSEQ call does not contain any inverted items.
44	Item referenced by FINDX or FINDSEQ call is not an inverted item.
45	DMSRETRN call without an available return address.
46	Password specified does not allow the intended procedural action.
47	Password not supplied for a password-secured database.
48	An area was open for update and the database lockout bit was set.
49	Either invalid argument in a procedural call, or the subschema definition of working storage does not match the definition in the program.
50	DBM call other than release with recovery made after a previous call was interrupted.
51	Attempt to open an area in shared mode after opening one or more in exclusive mode or vice versa.
52	Attempt to open an area with shared mode and the monitor version does not include enqueue/dequeue.
53	The users account authorization does not include use of enqueue/dequeue.
61	Attempt to store a group with items or sets omitted in the subschema.
62	Attempt to delete a group with sets or inverted items omitted in the subschema.
63	Attempt to link or delink a group that is not defined as an optional or manual member.
64	Attempt to use the FINDG procedural call without having all of the control items defined for the group or its owners.
65	Attempt to link a group in a sorted set without having all sort keys defined.
66	Attempt to modify a secondary index item or a sort control item and the invert group or the sorted set definition is omitted from the subschema.
67	Attempt to modify a data item that is a control item and one or more other control items are omitted.
68	Attempt to execute a FINDX or FINDSEQ procedure but the inverted option has been omitted for the item.

Table F-3. DBM Non-Data-Dependent Errors (cont.)

Error Number	Error Condition
69	A group has been retrieved that is not defined in the subschema.
70	Attempt to traverse a set that does not have the owner and all member groups defined.
71	The via set has not been defined for the referenced group.
72	Unable to store the new invert group occurrence for a modified secondary index item.
73	Unable to store an invert group occurrence for the secondary index item value in the group occurrence just stored.
80	The storage set has not been defined for a group.
81	A group occurrence has been retrieved that is of a different size than that specified by the subschema.
82	An operation was attempted on an indexed data group but the subschema does not contain a complete definition of the indexed area.
83	Group established to control secondary indexes is not a calc group.
84	The subschema does not define the invert group for a secondary index item.
85	Set control items are not defined correctly in the subschema.
86	Sort or random control items are not defined within the group by the subschema.
91	An area to be opened has not been assigned.
92	An area of the database is still unavailable after five attempts to open it.
93	The monitor has detected an illegal operation and returned to the trap routine.
94	The monitor has returned an error or abnormal code as the result of an I/O operation.
95	A page read from the database or subschema has an invalid checksum or the enciphering key presented by the user is not correct.
96	The page read from the database is not the correct page for the random block accessed.
97	Dynamic core memory is insufficient to load the subschema.
98	Dynamic memory available is insufficient to interface with Sort for a FINDSEQ procedure.
99-101	The memory space allocated for a User Argument table (99) Area definition table (100) Detail Pushdown list (101) has been exceeded.
121	Detail definition list is incorrect.
122	Group retrieved is not defined for set accessed.
123	Attempt to delink a group with set-next zero.

Table F-3. DBM Non-Data-Dependent Errors (cont.)

Error Number	Error Condition
124	Attempt to link a group with set-next zero.
125	Group specified by set-next cannot be retrieved.
126	Group just stored cannot be retrieved to complete set linkages.
127	Prior group of set cannot be retrieved.
128	Group specified by set-prior cannot be retrieved.
129	Unable to retrieve the main group while in the process of deleting the invert group occurrence for a secondary index item.
131	Unable to retrieve the invert group occurrence for the secondary index item of the current main group.
133	Sort processor has abnormally terminated while executing FINDSEQ sort.
134	The main group defined by a secondary index is not retrievable.
135	Unable to retrieve the group occurrence that was just created.
136	Invalid internal DBM argument.
137	Error has occurred in handling area owner group.

Table F-4. DMSINIT Error Messages

Message	Meaning
***ASSIGN CARD MISSING FOR area-name.	Area file identified by area-name was not assigned or was not properly assigned.
***F:DBnn NOT OUT OR INOUT FILE.	Function assigned for the F:DBnn is not OUT or INOUT.
***I/O ERROR F:SCHE - xx yy.	An I/O error return from the monitor occurred while processing the schema file - xx and yy are the major and minor codes returned by CP-V.
***I/O ERROR F:DBnn - xx yy.	An I/O error return from the monitor occurred while processing the area file assigned to F:DBnn – xx and yy are the major and minor status returned by CP-V.
***UNEXPECTED END OF FILE ON SI.	Period missing at end of statement, or additional input was expected.
***ILLEGAL RANGE.	Range specified was not within data pages of area.
***INCORRECT AREA NAME.	An area-name specified did not match any of the area names in the schema.
***RANGE NOT SPECIFIED FOR RE-INIT OR AN EXISTING AREA.	Range parameter is required to reinitialize an existing area.

Table F-4. DMSINIT Error Messages (cont.)

Message	Meaning
***PARTIAL RE-INIT OF INDEXED GROUP RANGE OR OVERFLOW RANGE NOT ALLOWED	If any pages in the indexed group's page range or in the overflow range are to be reinitialized, all must be reinitialized.
***ILLEGAL RE-INIT OF OVERFLOW-RANGE	Overflow range may not be reinitialized if indexed group range is not.
***SYNTAX ERROR	Missing equals sign or comma, misspelling of AREA or RANGE, etc.
***SCHEMA FILE IS BAD DBM ERROR CODE - xx	Error encountered in schema file – xx is the error code returned by the DBM.

Table F-5. DMSDUMP Error Messages

Message	Meaning
***SCHEMA FILE IS BAD, DBM ERROR CODE - xx	An error in the schema was detected by the DBM routines used to process it – xx is the DBM error code.
***INCORRECT AREA NAME	An area-name in a dump or print directive did not match any of the area names in the schema.
***SYNTAX ERROR	Missing equals sign, comma, etc.
***UNEXPECTED END OF FILE ON SI	Additional input was expected to complete an area, line, or group specification, or period was missing.
***INCORRECT DATA PAGE READ FROM area-name	Page read from database was not the desired page.
***ILLEGAL DIRECTIVE	Directive not PRINT or DUMP.
***ILLEGAL PASSWORD	Password not given or it was not a correct one.
***ILLEGAL RANGE	Range specified was not within the area.
***ASSIGN CARD MISSING FOR area-name	Area file identified by area-name was not assigned or not correctly assigned.
***I/O ERROR F:SCHE -- xx yy	An I/O error return from CP-V occurred while processing the schema file – xx and yy are the major and minor codes returned by CP-V.
***I/O ERROR, F:DBnn -- xx yy	An I/O error return from CP-V occurred on area file assigned to F:DBnn – xx and yy are the major and minor codes from CP-V.
***I/O ERROR F:DUMP -- xx yy	An I/O error return from CP-V occurred on sequential output file – xx and yy are the major and minor codes from CP-V.

Table F-5. DMSDUMP Error Messages (cont.)

Message	Meaning
***BAD LINE # OR GROUP LENGTH	Duplicate line numbers, zero group length, or invalid group length found on page being processed. Contents of page are printed in hexadecimal following this message.
***CHECKSUM ERROR OR PROPER CIPHER KEY REQUIRED	Either there was a checksum error, or the cipher key was not the proper key, or not in correct input order. The checksummed page is printed following this message. Checksum for the page is the last word printed.

Table F-6. DMSLOAD Error Messages

Message	Meaning
***SCHEMA FILE IS BAD, DBM ERROR CODE - xx	An error in the schema was detected by the DBM routines used to access it – xx is the error code returned by the DBM.
***INCORRECT AREA NAME	An area name in a LOAD, TAPE, or PRINT directive did not match any of the area names in the schema.
***INSUFFICIENT MEMORY FOR DMSLOAD	Not enough core space can be obtained for buffers.
***ASSIGN CARD MISSING FOR area-name	Area file identified by area-name was not assigned or not properly assigned.
***CIPHKEY/NEWCKEY NOT REQUIRED	Cipher key or new cipher key was specified for an area that is not enciphered.
***ILLEGAL RANGE	A range specified for an area did not correspond to the size of the area, was less than one, or was greater than the area size.
***UNEXPECTED END OF FILE ON SI	The input directive was incomplete, perhaps missing only the period.
***ILLEGAL DIRECTIVE	Directive identifier was not LOAD, TAPE, or PRINT.
***SYNTAX ERROR	Any of several format errors: missing comma, parenthesis, etc.
***I/O ERROR, F:SCHE -- xx yy	An I/O error return from the monitor occurred while processing the schema file – xx and yy are the major and minor codes returned by CP-V.
***I/O ERROR, AREA#=nn -- xx yy	An I/O error return from the monitor occurred while processing the area whose number is specified by nn – xx and yy are the major and minor codes returned by CP-V.

Table F-6. DMSLOAD Error Messages (cont.)

Message	Meaning
***I/O ERROR, F:LOAD -- xx yy	An I/O error return from the monitor occurred while reading the dump or journal file input – xx and yy are the major and minor codes returned by CP-V.
***I/O ERROR, F:DUMP -- xx yy	An I/O error return from the monitor occurred while writing the sequential file output – xx and yy are the major and minor codes returned by CP-V.
***BAD LINE # OR GROUP LENGTH	Duplicate line numbers, zero group length, or invalid group length found on a page being processed. Page in question is printed in hexadecimal following this message.
***CHECKSUM ERROR OR PROPER CIPHER KEY REQUIRED	Either there was a checksum error, or the cipher key was not the proper key, or not in correct input order. The checksummed page is printed following this message. The checksum for the page is the last word printed.
***WRONG INVENTORY PAGE -- xxxxxxxx	The page that was read in was not the inventory page expected. xxxxxxxx is the number of the desired page. The page read is printed in hexadecimal following this message.

Table F-7. DMSSUMS Error Messages

Message	Meaning
***CANNOT OPEN STATISTICS FILE	Statistics file was not assigned or did not exist. Processing is terminated.
***STATISTICS FILE WRONG FORM	The first record read from the statistics file was not a Job ID record. Processing terminates.
***I/O ERROR ON STATISTICS FILE	An error return from the monitor occurred while processing the statistics file. Processing is terminated.
***CANNOT OPEN SCHEMA FILE	The schema file does not exist, is not assigned, or is read-protected. Processing is terminated.
***I/O ERROR ON SI	An I/O error return from the monitor occurred while reading input. Processing is terminated.
***SCHEMA FILE IS BAD	An error return from the DBM routines used to process the schema occurred. Processing is terminated.
***UNRECOGNIZED SELECTION	Selection specification was not AREA, GROUP, or SET. Remaining selection input is scanned for errors but no statistics will be printed.
***INCORRECT AREA NAME	A specified area-name did not match any of the area names in the schema.

Table F-7. DMSSUMS Error Messages (cont.)

Message	Meaning
***SYNTAX ERROR	Missing comma, equals sign, etc. Remaining input is scanned.
***USE OF ALL MADE SPECIFIC SELECTION ILLEGAL	ALL may be used only once and no other selection is legal after ALL. Remaining input is scanned but no statistics are printed.
***UNEXPECTED END OF FILE ON SI	Missing period or partial selection was specified. Processing is terminated.
***INCORRECT GROUP NAME	Group-name specified was not in schema. Remaining input is scanned but no statistics are printed.
***INCORRECT SET NAME	Specified set-name was not in schema, remaining input is scanned but no statistics are output.

Table F-8. RPCL Error Messages

Message	Meaning
UNDEFINED ENTRY -- SKIPPED	DMSANLZ does not recognize the first word in the RPCL entry as a legal entry type. The entry is skipped.
ENTRY IS ILLEGAL DUPLICATE OR OUT OF ORDER -- ENTRY SKIPPED	The entry is not in the correct sequence within the RPCL or it is a duplicate of a previous entry. The entry is skipped.
SYNTAX ERROR -- SKIPPING TO NEXT ENTRY	The syntax of the entry violates the RPCL form for the entry. The entry is skipped.
CIPHER KEY EXCEEDS 4 CHARS IN LENGTH -- IGNORED	The cipher key for an area is greater than four characters. The cipher key is ignored.
FILL PERCENT IS NOT IN RANGE 1-100 OR IS GREATER THAN FILL PERCENT IN TARGET SCHEMA	The indexed group fill percent specified for an area in the RPCL is either an illegal percentage value or is greater than the fill percent for the area in the target schema DDL.
MAX NUMBER OF SERIAL NUMBERS EXCEEDED	Assembly parameter limit for maximum number of schema or area volume serial numbers has been exceeded. For A00, this limit is 10.
DATABASE AREA MUST BE ON RANDOM DEVICE	The volume serial number clause specified a device type for a database area that was other than DP.
AREA NOT EXCIPHERED -- CLAUSE MEANINGLESS	A cipher key has been specified for an area that is not enciphered. The clause is ignored.

Table F-8. RPCL Error Messages (cont.)

Message	Meaning
AREA HAS NO INDEXED DATA GROUP -- CLAUSE MEANINGLESS	An indexed group fill percent has been specified for an area that does not contain an indexed data group. The clause is ignored.
UNDEFINED AREA -- ENTRY SKIPPED	The area name specified in the RPCL area entry is not defined in the source or target schema. The entry is skipped.
TARGET AREA MUST BE DATABASE FORMAT	User attempted to specify a target area in DMSDUMP format.
FILL PERCENT MEANINGLESS FOR SOURCE AREA	An indexed group fill percent clause has been specified for a source area. The clause is ignored.
DUPLICATE CLAUSE -- ILLEGAL	A clause has been illegally duplicated. The clause is ignored.
PRIVACY KEY EXCEEDS 8 CHARS IN LENGTH -- IGNORED	The length of a schema privacy key is greater than eight characters. The key is ignored.
ILLEGAL NONNUMERIC LITERAL -- SKIPPING TO NEXT ENTRY	Either the closing apostrophe for a nonnumeric literal has been omitted or the user attempted continuation of a nonnumeric literal across multiple input units. The entry is skipped.
ILLEGAL CHARACTERS IN VOLUME NUMBER	A volume serial number contains characters other than A through Z and 0 through 9.
VOLUME SERIAL NUMBER HAS ILLEGAL NUMBER OF CHARS	The number of characters in a CP-V volume serial number is greater than four, or the number of characters in an ANS labeled tape volume serial number is not 6.
PASS NOT IN WEIGHT FACTOR SEQUENCE	A pass sequence subentry attempted to violate the rules for ordering load passes.
UNDEFINED GROUP	A group has been specified in a preserve or pass sequence subentry that is not defined in the target schema.
DUPLICATE GROUP SPECIFICATION	The user specified a group in more than one preserve or pass sequence subentry, or the same group appeared more than once in a single preserve or pass sequence subentry.

Table F-8. RPCL Error Messages (cont.)

Message	Meaning
UNDEFINED AREA	The area named in a load subentry is not defined in the target schema.
CAN'T PRESERVE CALC OR INDEXED GROUP	The user specified a group whose location mode is calc or indexed in a preserve subentry.
ATTEMPT TO LOAD GROUP group-name AHEAD OF ITS STORAGE MASTER	A pass sequence subentry specified the named group to be loaded prior to loading its storage set owner or in the same load pass as its storage set owner.
CAN'T PRESERVE GROUP THAT SHARES PAGE RANGE OF INDEXED DATA GROUP	A group which resides in the same page range as an indexed data group has been specified in a preserve subentry.

Table F-9. DMSREST Error Messages

Message	Meaning
DCB F:WFnn ASSIGNED TO UNRECOGNIZED DEVICE	DCB F:WFnn is assigned to a device type that is not recognized by DMSREST. The device types recognized are DP, 7T, and 9T.
DCB F:WFnn NOT ASSIGNED TO FILE, LABEL, OR ANSLBL	DCB F:WFnn has an explicit or default device assignment. Assignment must be to FILE, LABEL, or ANSLBL.
SORT ERROR n	DMSREST has linked to the XEROX SORT program and the sort has completed with error code n. See the Xerox Sort and Merge Reference Manual for an explanation of error codes returned by SORT in register 6.
I/O ERROR AT LOC nnn IN name CODE nn SUBCODE nn ON dcbname	A error or abnormal return was made by the monitor to the I/O CAL near location "nnn" in DMSREST module "name". The message includes the error code and subcode along with the DCB referenced.
nK WORDS ADDITIONAL MEMORY REQUIRED	Insufficient memory is available to DMSREST. A memory increment of at least nK words is required.

Table F-9. DMSREST Error Messages (cont.)

Message	Meaning
ERROR nn ... FILE INCONSISTENCY IN filename	DMSREST has detected a file error, where code nn is: 21 - file I/O error 22 - file I/O abnormal 23 - number of blocks processed do not equal number of blocks required 24 - block or database page number transmitted is not the one required 25 - checksum or encipher error in block or database page 26 - RPCC recovered from wrong checkpoint file 27 - number of records processed do not equal the number of record required 28 - invalid record length 29 - invalid origin for a block or database page 30 - illegal line number present in database page 31 - available space invalid in database page 32 - duplicate line number present in database page 33 - too many SNs present 34 - SN lost by monitor
ERROR nn ... PROGRAM FAULT ... REPORT TO XEROX ANALYST	A program error nn has occurred from which recovery is not possible.

Table F-10. APL/EDMS Errors

Error Code	Message Text	Comments
501	INSUFFICIENT SUBSCHEMA FOR APL/EDMS OR FILE NAMED IN DMSSUB NOT SUBSCHEMA	If file named in DMSSUB call is indeed a subschema, then it was created by a DMSFDP version before C00.
502	CHECKSUM ERROR IN SUBSCHEMA FILE	
503	DMS CALL AFTER FATAL ERROR nnn HAS OCCURRED	nnn is error code listed in Table F-3.
504	I/O ERROR READING SUBSCHEMA FILE	The monitor error code is available via APL ERRX intrinsic.
505	FIRST EDMS CALL NOT 'DMSSUB'	
506	NO DCB AVAILABLE FOR AREA TO BE OPENED	
508	EDMS PUBLIC LIBRARY IS NOT IN ':P22'	
509	ATTEMPT TO EXECUTE DMSEND OR DMSSUB WITH ACTIVE AREAS	
510	ILLEGAL DIGIT IN ZONED OR PACKED ITEM VALUE	
511	VALUE IN NUMERIC OR PACKED ITEM WILL NOT FIT IN FLOATING LONG NUMBER	
512	ARGUMENT NAME NOT IN NAME TABLE	
513	NON-UNIQUE ITEM REFERENCE WAS NOT QUALIFIED	
514	UNABLE TO ASSOCIATE ITEM-NAME WITH QUALIFIER	
515	TRUNCATION	
516	ATTEMPT TO ASSIGN NEGATIVE VALUE TO UNSIGNED ITEM	
517	DMS CALL WITH NO OPEN AREAS	
518	ATTEMPT TO EXECUTE OPEN CALL WITH ACTIVE AREAS	
519	CAN'T GIVE YOU ERROR CONTROL—ERROR FUNC IS NOT NYLADIC, NO-RESULT	
520	LOAD, CLEAR, OFF, CONTINUE AND SAVE COMMANDS NOT LEGAL WITH ACTIVE AREAS	

APPENDIX G. DATA VALIDATION

EDMS provides for the validation of data item values when they are stored or modified in the database. Through clauses in the Schema DDL, validation may be specified against a picture of the item and a range of values for the item.

If picture validation is requested, the value of each character in the item is compared to a set of allowed values for the corresponding picture character. If the values do not agree, a data dependent error is returned. The picture character and the allowed values for EBCDIC items are:

Picture	Values
9	Hexadecimal F0-F9 (numeric values only)
A	Hexadecimal C1-C9, D1-D9 and E2-E9 (alphabetic values) and hexadecimal 40 (space).
X	Hexadecimal 00-FF (all values).

If the item type is signed numeric, the allowed values for the low order character position are A0-A9, B0-B9, C0-C9, D0-D9, E0-E9 and F0-F9. Packed decimal values are checked to ensure that each half byte contains a valid numeric value, i.e., 0-9. If the number of characters in the item is even, the value of the first half byte of the item must be zero. The half byte for the sign character is checked for a hexadecimal value in the range A through F, inclusive.

If range validation is requested, the value of the data item is compared to the converted values for the literals supplied in the check clause of the DDL. The item value must be equal to either literal value or it must be greater than the low literal and less than the high literal. Validation against a single value may be accomplished by using the same literal for the low and high values in the DDL check clause. The user is cautioned against using this approach for the extremes in floating point short and floating point long values. Different programming languages may convert the same literal to different floating point representations. The File Definition Processor converts literals in the DDL by using the same routines as the Xerox COBOL compiler. If a program is written in FORTRAN or Meta-Symbol, literals supplied for item values may not be converted to identical floating point representations. The DBM may thus return an error condition if a range of values was not specified in the check clause.

When comparing signed numeric or numeric EBCDIC item values to the literals, the DBM will ignore the first half byte of each character position except the low order character. In the low order character position hexadecimal values A, C, E or F are considered as a positive sign and values B or D as a negative sign.

The DBM uses the decimal instructions of the hardware for comparison of packed decimal values. Thus, sign values A, C, E and F are considered positive and values B and D as negative.

In the comparison of data items to range literals, no check is made to ensure that the characters in the item are valid characters for the item type. This is only done for picture validation.

APPENDIX H. ENQUEUE/DEQUEUE

The DBM uses the enqueue/dequeue function of the CP-V operating system to control the interaction of programs concurrently accessing an area of the database. The enqueue function provides for control of a global resource (the database area) and/or an element of that resource (a page of the area) at two levels, shared use or exclusive use.

The DBM issues an enqueue request for shared use of an element just prior to the read of each page from the area. If the request is successful the page is locked to the program for shared use. If the request is not successful, the program is suspended by the operating system until the request can be satisfied. When an element is enqueued for shared use, other programs may also enqueue the element for shared use (i.e., can read the page).

The DBM issues an enqueue request for exclusive use of an element just prior to modifying the data page in the DBM buffer. If the request is successful the page is locked to the program for exclusive use. If the request is not successful the program is suspended until the request can be satisfied. The operating system will not allow an enqueue request for exclusive use if some other program has that element enqueued for shared use. Once an element is enqueued for exclusive use no other program may enqueue that element for shared use.

When the user program issues a DMSRLSE procedural call or closes the last open area, the DBM will dequeue all elements locked for that program. Other programs that may have been suspended because of conflicting enqueue requests may then be placed back in execution by the operating system.

Through DBM use of enqueue/dequeue, concurrently executing programs are protected from interfering with each other. When the DBM sends a page and returns data or a set position to the program, that data or set position cannot be changed by another program until the reading program explicitly releases the page. When a program has updated a page no other program may read the modified data or set pointers until the updating program releases the page.

As previously stated, each DBM enqueue request is for a global resource (the area) and an element (a page) of that resource. The area is defined in the enqueue request by a 3-byte hashed value of the area name and the account under which it exists. The page is identified by the 20-bit EDMS page number. A hashed value is used in place of the full area name and account number to reduce the space and time required in accessing the operating system enqueue tables. When a hashing technique is used there is a possibility that the values derived from two or more area names may result in duplicates. If this should occur the result would be an overprotection of the programs accessing the two areas. The user should also be aware of the conflicts that may result if the enqueue/dequeue function is used in a program to protect a resource other than a database area and that resource has the same resource name as a database area.

A program has been included on the EDMS release tape to enable the user to identify potential conflicts in resource names. This program, named Hash, reads area name/account number pairs through M:SI DCB. The format is AREA-NAME.ACCOUNT-NUMBER. Each pair must begin on a new input record (i.e., card, edit line etc.). Output is through the M:LO DCB and is the 3-byte hash value derived from the AREA-NAME/ACCOUNT-NUMBER pair. The hash value is displayed as six hexadecimal characters.

Following is a sample run of the program.

```
!BUILD HASHVALUE
   1.000 AREA-1.693AA3JP
   2.000 MFG-DB-01.0131
   3.000 OUR-DATABASE-AREA-01.MYACCT
   4.000

!SET M:SI DC/HASHVALUE

!HASH.
AREA-1.693AA3JP
HASH VALUE = 2B4215
MFG-DB-01.0131
HASH VALUE = 69DF64
OUR-DATABASE-AREA-01.MYACCT
HASH VALUE = 43FDFB
```

APPENDIX I. DMSREST PROCESS FLOW

Phase 1: Unload

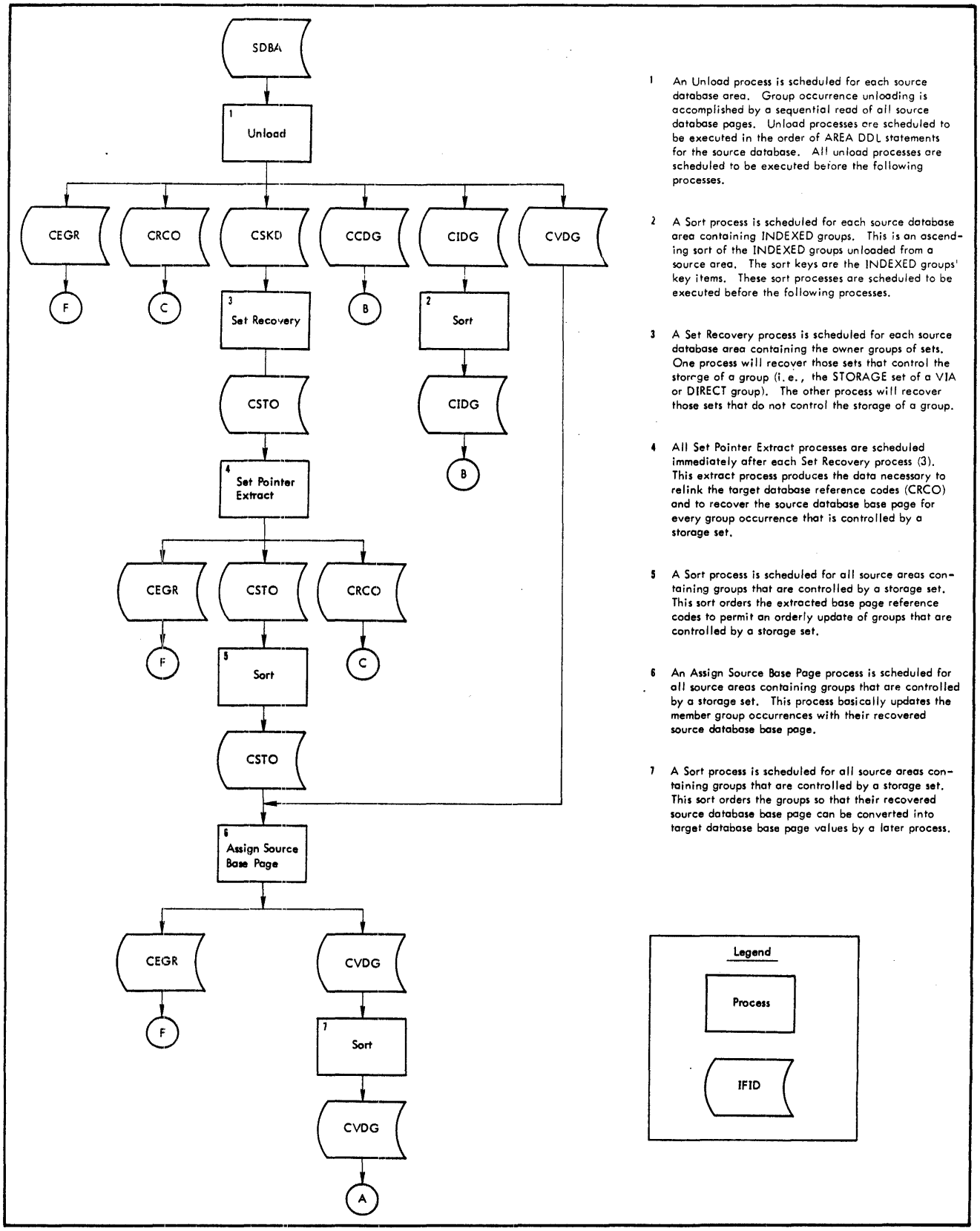
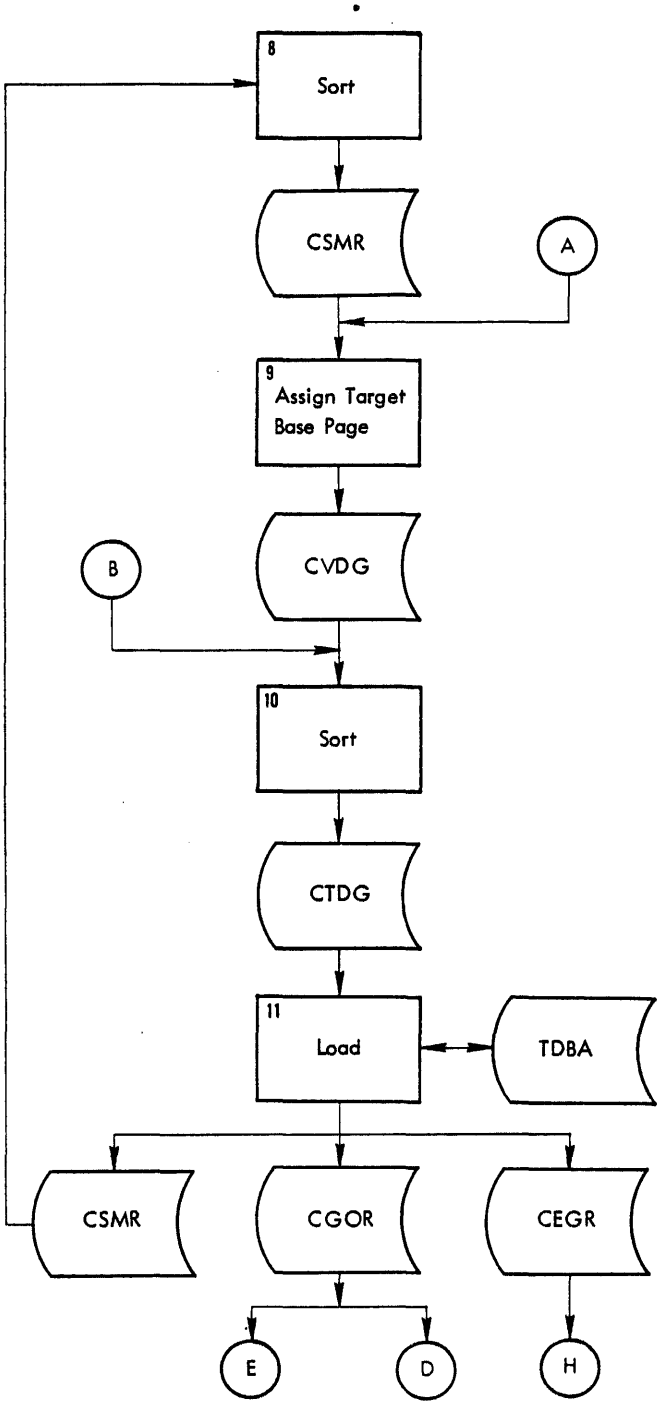


Figure I-1. DMSREST Flow Diagram

Phase 2: Load



8 A Sort process is scheduled prior to a load pass for groups that are controlled by a storage set. This sort orders the source and target database base page so that the target database base page may be assigned to member group occurrences by the next scheduled process.

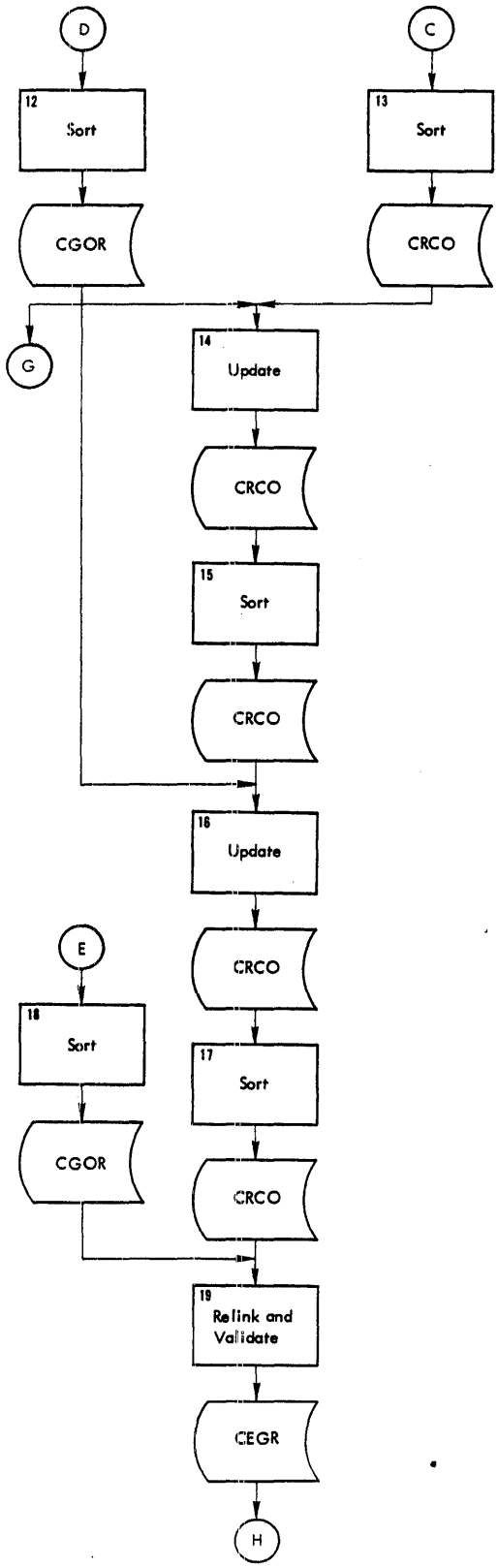
9 An Assign Target Base Page process is scheduled prior to a load pass for groups that are controlled by a storage set. This process basically updates these group occurrences with their actual target database base page prior to loading them.

10 A Sort process is scheduled for all group occurrences that are to be loaded in the next load pass. This sort orders group occurrences according to the data loading sequence criteria.

11 A Load process is scheduled for each load pass necessary to meet the DMSANLZ default or user influenced data loading sequence. Group occurrence loading is accomplished by a sequential update of some target database pages. Any group that controls the storage of another group through a storage set, causes source and target base page reference codes to be saved (CSMR) for later use in loading such storage controlled group occurrences. The source and target database reference codes of each loaded group occurrence are saved (CGOR) to permit relinking the target database sets and inverted group occurrences.

Figure I-1. DMSREST Flow Diagram (cont.)

Phase 3: Relink and Validate



12 A Sort process is scheduled after the last scheduled load process (11). This sort orders all source and target database reference codes of the loaded group occurrences so that the set pointer and inverted group source database reference codes may be assigned their equivalent target database reference code values.

13 A Sort process is scheduled to order all set pointer and inverted group source database reference codes. This sort orders the source database location of these reference codes so that they may be assigned their target database location by the next process.

14 An Update process is scheduled and assigns the target database location to each source database set pointer and inverted group reference code.

15 A Sort process is scheduled to order all source database set pointer and inverted group reference codes on their source database values. This sort permits the assignment of target database reference codes to these set pointer and inverted group source database reference codes by the next process.

16 An Update process is scheduled and assigns the target database reference code values of each set pointer and inverted group occurrence.

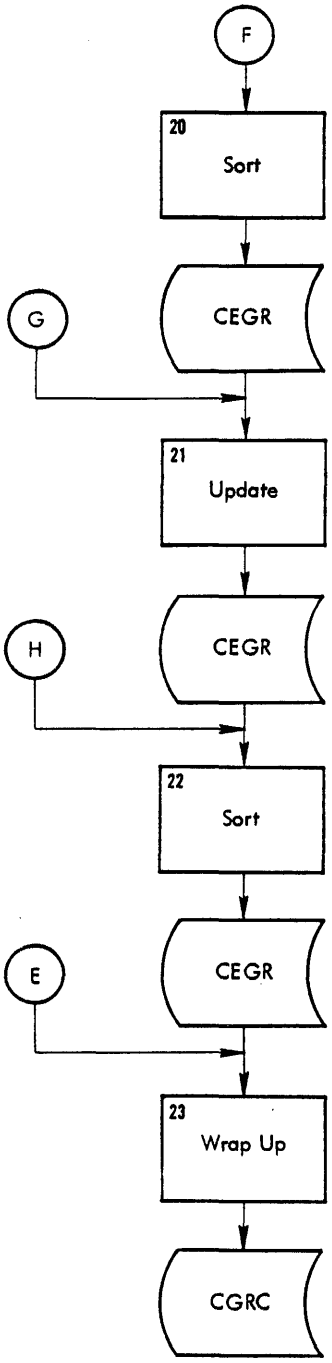
17 A Sort process is scheduled to order all set pointer and inverted group occurrence reference codes on their target database locations.

18 A Sort process is scheduled to order all locations of loaded group occurrences.

19 A Relink and Validate process is scheduled for each target database area. This process relinks all set pointers and inverted groups by substituting target reference codes for their source reference codes. The set pointers are validated to assure that this substitution is correct. All target group occurrences are validated to verify that no source group occurrences were lost and that no spurious target group occurrences were loaded into the target database area by DMSREST.

Figure I-1. DMSREST Flow Diagram (cont.)

Phase 4: Wrap Up



20 A Sort process is scheduled to order all DMSREST detected source database errors on their source database locations. This sort permits the assignment of target database locations to these errors by the next process.

21 An Update process is scheduled and assigns the target database location to each source database error detected by DMSREST.

22 A Sort process is scheduled to order all DMSREST detected source and target database errors on their target database locations.

23 A Wrap Up process is scheduled to report all source and target database errors detected by DMSREST, and to produce a correspondence file of all source and target database reference codes.

Figure I-1. DMSREST Flow Diagram (cont.)

APPENDIX J. SAMPLE DATABASE RESTRUCTURING

This appendix illustrates various aspects of the Restructuring Subsystem and the operation of DMSANLZ and DMSREST (see Figures J-1 through J-12). The database being restructured is the sample database of Figure 1 in this manual. The source database schema DDL is presented in Figure C-1; the target database schema is that shown in Figure J-1. The DMSANLZ and DMSREST outputs and an IFID file catalog listing are provided.

```
***200*** EXTENDED DMS FILE DEFINITION PROCESSOR -- VERSION B00.
.
***210*** EXTENDED DMS SCHEMA DDL.
   1:  SCHEMA NAME IS SAMPLESCHEMAMOD; PRIVACY LOCK FOR
   2:        EXTRACT IS 'EXLOCK'; PASSWORD IS 'PASWRD1'
   3:        RETRIEVE KEYS ARE 1,17    UPDATE KEY IS 231,247
   4:        PASSWORD IS 'PASWRD2' RETRIEVE KEY IS 1,17
   5:        PASSWORD IS 'PASWRD3'.
   6:  AREA NAME IS AREA-1 CONTAINS 100 PAGES; NUMBER IS 1
   7:        ;INVENTORY 80
   8:        ;CHECKSUM IS REQUIRED; JOURNAL IS NOT
   9:        REQUIRED; ENCIPHERING IS NOT REQUIRED.
  10:  AREA NAME IS AREA-2 CONTAINS 100 PAGES; NUMBER
  11:        IS 2; INVENTORY
  12:        PERCENT IS 80; JOURNAL IS NOT REQUIRED.
  13:  GROUP NAME IS GROUP-1 WITHIN AREA-1; NUMBER IS
  14:        100; LOCATION MODE IS DIRECT; PRIVACY LOCK
  15:        FOR RETRIEVE IS 1; PRIVACY LOCK FOR UPDATE IS
  16:        231; STATISTICS ARE REQUIRED.
  17:  GROUP NAME IS GROUP-2 WITHI' AREA-1
  18:        NUMBER IS 200; LOCATION MODE IS VIA
  19:        SET-A SET PRIVACY LOCK FOR RETRIEVE IS 1
  20:        PRIVACY LOCK FOR UPDATE IS 231.
  21:  ITEM-21; PICTURE A(16) TYPE IS CHARACTER
  22:        PRIVACY LOCK FOR RETRIEVE IS 17; PRIVACY
  23:        LOCK FOR UPDATE IS 231.
  24:  ITEM-22 TYPE IS BINARY; OCCURS 4.
  25:  ITEM-23; TYPE IS FLOATING LONG.
  26:  GROUP NAME IS GROUP-3 WITHIN AREA-2 RANGE 1 THRU 50 LOCATION MODE
  27:        IS CALC USING ITEM-32 DUPLICATES ARE ALLOWED;
  28:        NUMBER IS 300; PRIVACY LOCK FOR RETRIEVE
  29:        IS 17 PRIVACY LOCK FOR UPDATE IS 247.
  30:  ITEM-31 PICTURE X(31) OCCURS 4 TIMES.
  31:  ITEM-32; TYPE IS CHARACTER,31.
  32:  GROUP NAME IS GROUP-4 WITHIN AREA-2 RANGE IS 51 THRU 100
  33:        NUMBER IS 400; LOCATION MODE IS CALC USING
  34:        ITEM-41 DUPLICATES NOT ALLOWED PRIVACY LOCK FOR RETRIEVE IS 17
  35:        PRIVACY LOCK FOR UPDATE IS 247.
  36:  ITEM-41 PICTURE IS 99V99.
  37:  ITEM-42 PICTURE IS AA9(4)A PRIVACY LOCK FOR RETRIEVE
  38:         IS 17.
  39:  ITEM-43 TYPE IS CHARACTER,4.
  40:  ITEM-44 TYPE IS BINARY.
  41:  SET NAME IS SET-A; OWNER IS GROUP-1; ORDER IS FIRST.
  42:  MEMBER IS GROUP-2
  43:        ;INCLUSION IS AUTOMATIC SET OCCURRENCE SELECTION
  44:        IS LOCATION MODE OF OWNER.
  45:  SET NAME IS SET-B; OWNER IS GROUP-2
  46:        ORDER IS NEXT; STATISTICS ARE REQUIRED.
  47:  MEMBER IS GROUP-3 INCLUSION IS AUTOMATIC
  48:        ;LINKED TO OWNER; SET OCCURRENCE SELECTION
  49:        IS THRU CURRENT OF SET.
  50:  SET NAME IS SET-C; ORDER IS NEXT
  51:        ;OWNER IS GROUP-2; LINKED TO PRIOR
  52:        ;STATISTICS ARE REQUIRED.
  53:  MEMBER IS GROUP-4 INCLUSION IS MANUAL
  54:        SELECTION IS THRU CURRENT OF SET.
  55:  SET NAME IS SET-D; OWNER IS GROUP-3
  56:        ; ORDER IS SORTED; STATISTICS ARE
  57:        REQUIRED.
```

Figure J-1. Schema DDL Listing for Sample Target Database

```
  58:  MEMBER IS GROUP-4 INCLUSION IS AUTOMATIC
  59:       ;LINKED TO OWNER
  60:       ;SET OCCURRENCE SELECTION IS THRU LOCATION
  61:       MODE OF OWNER; ASCENDING RANGE KEY IS
  62:       ITEM-41 DUPLICATES ARE NOT ALLOWED.
  63:  END.

***207*** SCHEMA CONTAINS 0005 PAGES.

***209*** THERE WERE 0000 DIAGNOSTIC MESSAGES.


                    STORAGE REQUIREMENT SUMMARY

   AREA     DATA PAGES       INDEX PAGES     INVENTORY PGS    TOTAL PAGES

    01       0000100          0000000          0000001          0000101
    02       0000100          0000000          0000001          0000101


***201*** SCHEMA GENERATION COMPLETE.
```

Figure J-1. Schema DDL Listing for Sample Target Database (cont.)

```
00:30 JUL 18,'74     DMSANLZ FILE,RELEASE

CURRENT OPTIONS:     LOG

                     WORK=          8

                     SORT=         17

                     BUFFERS=       2

                     BLOCK=      2048

                     FILE

                     RELEASE

                     CHECKSUM

                     CIPHER
```

Figure J-2. DMSANLZ Control Command Option Listing for Sample Schema Analysis

```
00:30 JUL 18,'74      * * *   P R O C E S S   C O N T R O L   L A N G U A G E
   1:  SOURCE SCHEMA IS SAMPLESCHEMA PRIVACY EXTRACT 'EXLOCK'.
   2:  TARGET SCHEMA IS SAMPLESCHEMAMOD PRIVACY EXTRACT 'EXLOCK'.
   3:  SOURCE AREA AREA-1 DMSDUMP SAMPDBDUMP.
   4:  SOURCE AREA AREA-2 DMSDUMP SAMPDBDUMP.
   5:  END.

.THERE WERE 000 DIAGNOSTIC MESSAGES

RPCL PROCESSING COMPLETE
```

Figure J-3. DMSANLZ RPCL Listing for Sample Schema Analysis

```
00:30 JUL 18,'74            * * *  C O M P O N E N T  A N A L Y S I S  * * *
          PASSWORDS AND/OR ACCESS CODES CHANGED
          INVENTORY PERCENT OF AREA AREA=1 CHANGED
          CHECKSUM OPTION OF AREA AREA=1 CHANGED
          ACCESS KEYS OF GROUP GROUP=2 CHANGED
          PAGE RANGE OF GROUP GROUP=2 CHANGED
          INVENTORY PERCENT OF AREA AREA=2 CHANGED
          DATA PAGES OF AREA AREA=2 CHANGED
          ACCESS KEYS OF GROUP GROUP=4 CHANGED
          PAGE RANGE OF GROUP GROUP=4 CHANGED
          ACCESS KEYS OF ITEM ITEM=41 IN GROUP GROUP=4 CHANGED
          ACCESS KEYS OF ITEM ITEM=42 IN GROUP GROUP=4 CHANGED

          THERE WERE   11 LEGAL CHANGES

          THERE WERE    0 ILLEGAL CHANGES
```

Figure J-4. DMSANLZ Source and Target Schema Component Analysis Listing for Sample Schema Analysis

```
00:30 JUL 18,'74            * * *  D A T A  L O A D  S E Q U E N C E  * * *
 PASS  STEP      AREA                        GROUPS LOADED
    1    13   AREA=1                      GROUP=1
    2    17   AREA=1                      GROUP=2
    3    19   AREA=2                      GROUP=3                         GROUP=4
```

Figure J-5. DMSANLZ Target Database Load Sequence Listing for Sample Schema Analysis

```
00:30 JUL 18,'74    * * *  S C H E D U L E D  P R O C E S S  S E Q U E N C E  * * *
 STEP:    2     PROCESS: UNLD -- UNLOAD SOURCE AREA SAMPDBDUMP
        IFID      GROUPS
        CCDG  GROUP=3                     GROUP=4

     INPUT FILES
        IFID  FILE NAME                   STEP CREATED BY  DEVICE    CIPHERED  ACCOUNT   VOLUMES
        SDBA  SAMPDBDUMP                       DUMP        PUBLIC       NO

     OUTPUT FILES
        IFID  FILE NAME                        DCB         DEVICE    CIPHERED
        CCDG  CCDG=0002-                    FIWF02         PUBLIC       NO
        CSKD  CSKD=0002-                    FIWF04         PUBLIC       NO
        CRCB  CRCB=0002-                    FIWF05         PUBLIC       NO
        CEGR  CEGR=0002-                    FIWF06         PUBLIC       NO
```

Figure J-6. DMSANLZ Scheduled Process Sequence Listing for Sample Schema Analysis

```
00:30 JUL 18,'74     * * *  S C H E D U L E D   P R O C E S S   S E Q U E N C E  * * *                23

 STEP:   18      PROCESS: LOAD -- LOAD GROUP OCCURRENCES INTO TARGET AREA AREA=2

                   GROUPS

                GROUP=3                          GROUP=4

      INPUT FILES
         IFID  FILE NAME                    STEP CREATED BY  DEVICE    CIPHERED  ACCOUNT    VOLUMES

         CTDG  CTDG=0018=                          18         PUBLIC      NO

      OUTPUT FILES
         IFID  FILE NAME                       DCB           DEVICE    CIPHERED

         CSMR  CSMR=0019=                    FIWF01          PUBLIC      NO
         CGBR  CGBR=0019=                    FIWF04          PUBLIC      NO
         CEGR  CEGR=0019=                    FIWF06          PUBLIC      NO

      INOUT FILES
         IFID  FILE NAME                    STEP CREATED BY  DEVICE    CIPHERED  ACCOUNT    VOLUMES

         TDBA  AREA=2                              USER       PUBLIC      NO
```

Figure J-6. DMSANLZ Scheduled Process Sequence Listing for Sample Schema Analysis (cont.)

```
00:30 JUL 18,'74          * * *  S C H E D U L E D   F I L E   U S A G E  * * *                      37

  IFID   FILE NAME                    DEVICE    CIPH   BYTS   STEP CREATED BY    STEPS USED BY    VOLUMES

  SDBA   SAMPDBDUMP                   PUBLIC     NO    2048        DUMP          1 - UNLD
  CCDG   CCDG=0001=                   PUBLIC     NO      32      1 - UNLD        12 - SRTS
  CVDG   CVDG=0001=                   PUBLIC     NO      84      1 - UNLD         6 - ASBP
  CSKD   CSKD=0001=                   PUBLIC     NO      24      1 - UNLD         3 - PPSB
                                                                                  8 - PPSB
  CRCB   CRCB=0001=                   PUBLIC     NO      24      1 - UNLD        21 - SRTA
  CEGR   CEGR=0001=                   PUBLIC     NO      32      1 - UNLD        29 - SRTA
  SDBA   SAMPDBDUMP                   PUBLIC     NO    2048        DUMP          2 - UNLD
  CCDG   CCDG=0002=                   PUBLIC     NO     195      2 - UNLD        18 - SRTS
  CSKD   CSKD=0002=                   PUBLIC     NO      24      2 - UNLD         8 - PPSB
                                                                                 10 - PPSB
  CRCB   CRCB=0002=                   PUBLIC     NO      24      2 - UNLD        21 - SRTA
  CEGR   CEGR=0002=                   PUBLIC     NO      32      2 - UNLD        29 - SRTA
  CSTB   CSTB=0003=                   PUBLIC     NO      24      3 - PPSB         4 - XSET
  CSTB   CSTB=0004=                   PUBLIC     NO      24      4 - XSET         5 - SRTA
  CRCB   CRCB=0004=                   PUBLIC     NO      24      4 - XSET        21 - SRTA
  CEGR   CEGR=0004=                   PUBLIC     NO      32      4 - XSET        29 - SRTA
  CSTB   CSTB=0005=                   PUBLIC     NO      24      5 - SRTA         6 - ASBP
```

Figure J-7. DMSANLZ Scheduled File Usage Listing for Sample Schema Analysis

```
00:33 JUL 18,'74     DMSREST
CURRENT OPTIONS:     LOG
                     WORK=          8
                     SORT=         17
                     BUFFERS=       2
                     BLOCK=      2048
                     FILE
                     RELEASE
                     CHECKSUM
                     CIPHER
```

Figure J-8. DMSREST Control Command Option Listing for Sample Restructuring

```
00:33 JUL 18,'74    STEP 2          UNLD -- UNLOAD GROUP OCCURRENCES FROM SOURCE AREA                    3

IFID   FILE     DCB &     I/O &    ORGAN.  FILE NAME IN RPCC CATALOG        BYTE SIZE OF:   NUMBER OF FILE:
       STATUS   TYPE      DEVICE   ACCESS     & PROTECTION ATTRIBUTES      BLOCK  RECORD    BLOCKS   RECORDS

SDBA   SAVED    F:WF08    INPUT    RANDOM  SAMPDB01=P                      2076    2048        50        90
                FILE      PUBLIC   DIRECT     CHECKSUMMED

CCDG   SAVED    F:WF02    OUTPUT   CONSEC  CCDG=0002=286593                1972     195         8        90
                FILE      PUBLIC   SEQUEN     CHECKSUMMED
                                              ACCOUNT K19113JP

CSKD   SAVED    F:WF04    OUTPUT   CONSEC  CSKD=0002=286593                2048      24         3       180
                FILE      PUBLIC   SEQUEN     CHECKSUMMED
                                              ACCOUNT K19113JP

CRC8   UNUSED   F:WF05    OUTPUT   CONSEC  CRC8=0002=286593                2048      24         0         0

CEGR   UNUSED   F:WF06    OUTPUT   CONSEC  CEGR=0002=286593                2024      32         0         0

ELAPSED STEP TIME   0100   STEP EXECUTION TIME   0.0372   STEP SERVICE TIME   0.0312        61       360


00:35 JUL 18,'74    STEP 19         LOAD -- LOAD GROUP OCCURRENCES INTO TARGET AREA                      23

IFID   FILE     DCB &     I/O &    ORGAN.  FILE NAME IN RPCC CATALOG        BYTE SIZE OF:   NUMBER OF FILE:
       STATUS   TYPE      DEVICE   ACCESS     & PROTECTION ATTRIBUTES      BLOCK  RECORD    BLOCKS   RECORDS

CTDG   SAVED    F:WF09    INPUT    CONSEC  CTDG=0019=286595                1972     195         8        90
                FILE      PUBLIC   SEQUEN     CHECKSUMMED
                                              ACCOUNT K19113JP

CSMR   UNUSED   F:WF01    OUTPUT   CONSEC  CSMR=0019=286595                2048      12         0         0

CGBR   SAVED    F:WF04    OUTPUT   CONSEC  CGBR=0019=286595                2048      12         1        90
                FILE      PUBLIC   SEQUEN     CHECKSUMMED
                                              ACCOUNT K19113JP

CEGR   UNUSED   F:WF06    OUTPUT   CONSEC  CEGR=0019=286595                2024      32         0         0

TDBA   SAVED    F:WF08    INOUT    RANDOM  AREA=2                          2048    2048       100         0
                FILE      PUBLIC   DIRECT     CHECKSUMMED

ELAPSED STEP TIME   0100   STEP EXECUTION TIME   0.0263   STEP SERVICE TIME   0.0321       109       180
```

Figure J-9. DMSREST Executed Scheduled Process Sequence Listing for Sample Restructuring

```
00:38 JUL 18,'74    * * *  R P C C   C A T A L O G U E D   F I L E S   L I S T I N G  * * *  I      4

CVDG  RELEASE  F:WF03  OUTPUT  CONSEC  CVDG=0006=286593                2028    84        2        30
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CVDG  RELEASE  F:WF03  OUTPUT  CONSEC  CVDG=0007=286593                2028    84        2        30
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CST8  RELEASE  F:WF04  OUTPUT  CONSEC  CST8=0008=286594                2048    24        2       150
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CRC8  RELEASE  F:WF05  OUTPUT  CONSEC  CRC8=0009=286594                2048    24        4       270
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CST8  RELEASE  F:WF04  OUTPUT  CONSEC  CST8=0010=286594                2048    24        2        90
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CRC8  RELEASE  F:WF05  OUTPUT  CONSEC  CRC8=0011=286594                2048    24        2       120
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CEGR  RELEASE  F:WF06  OUTPUT  CONSEC  CEGR=0011=286594                2024    32        1         1
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

CTDG  SAVED    F:WF07  OUTPUT  CONSEC  CTDG=0012=286594                2028    32        1        10
               FILE    PUBLIC  SEQUEN     CHECKSUMMED

TDBA  SAVED    F:WF08  INOUT   RANDOM  AREA=1                         2048  2048      100        40
               FILE    PUBLIC  DIRECT     CHECKSUMMED
```

Figure J-10. DMSREST RPCC Cataloged Files Listing for Sample Restructuring (Listing Produced as the Result of CATALOG Keyin)

```
GROUP GROUP=4 CONTAINS BAD HEAD POINTER AT BYTE   39 FOR SET SET=0 -- SOURCE HEAD POINTER WAS
     02002801 SHOULD HAVE BEEN 02002901 -- SOURCE GROUP REFCODE WAS 02001102
     -- TARGET GROUP REFCODE IS 02005901

THERE WERE      1 ERRORS LOGGED
```

Figure J-11. DMSREST Error Summary Listing for Sample Restructuring

```
41:
DMSREST - WILL ACCEPT OPERATOR INTERRUPTS

41:
DMSREST - UNLD 0001
00:33

41:
DMSREST - UNLD 0002
41:
DMSREST - PPSR 0003
!INT,41

41:
DMSREST - XSET 0004
41:
DMSREST - INTERRUPTED BY OPERATOR, YOU MAY INPUT:
41:
   IGNORE   - TO IGNORE YOUR INTERRUPT
41:
   PAUSE    - TO PAUSE DMSREST AFTER CURRENT STEP
41:
   BREAK    - TO TERMINATE DMSREST AFTER CURRENT STEP
41:
   GO       - TO CANCEL YOUR PRIOR 'PAUSE' OR 'BREAK'
41:
   CATALOG  - TO LIST ALL FILES AFTER CURRENT STEP
41:
   QUIT     - TO IMMEDIATELY TERMINATE DMSREST
41:
DMSREST - YOUR INPUT IGNORE

00:34

41:
DMSREST - YOU'RE IGNORED
41:
DMSREST - SRTA 0005
```

Figure J-12. DMSREST Operator Console Listing for Sample Restructuring

APPENDIX K. DMSREST SEQUENTIAL FILE FORMATS

This appendix defines the format of the sequential file generated by DMSREST that may be accessed by a user. This file is the Conveyed Group's Reference Code (CGRC) file that is produced by the wrap-up process of DMSREST. It contains a record for each group occurrence conveyed from the source database to the target database. The record includes the reference code for the group in the source database and in the target database.

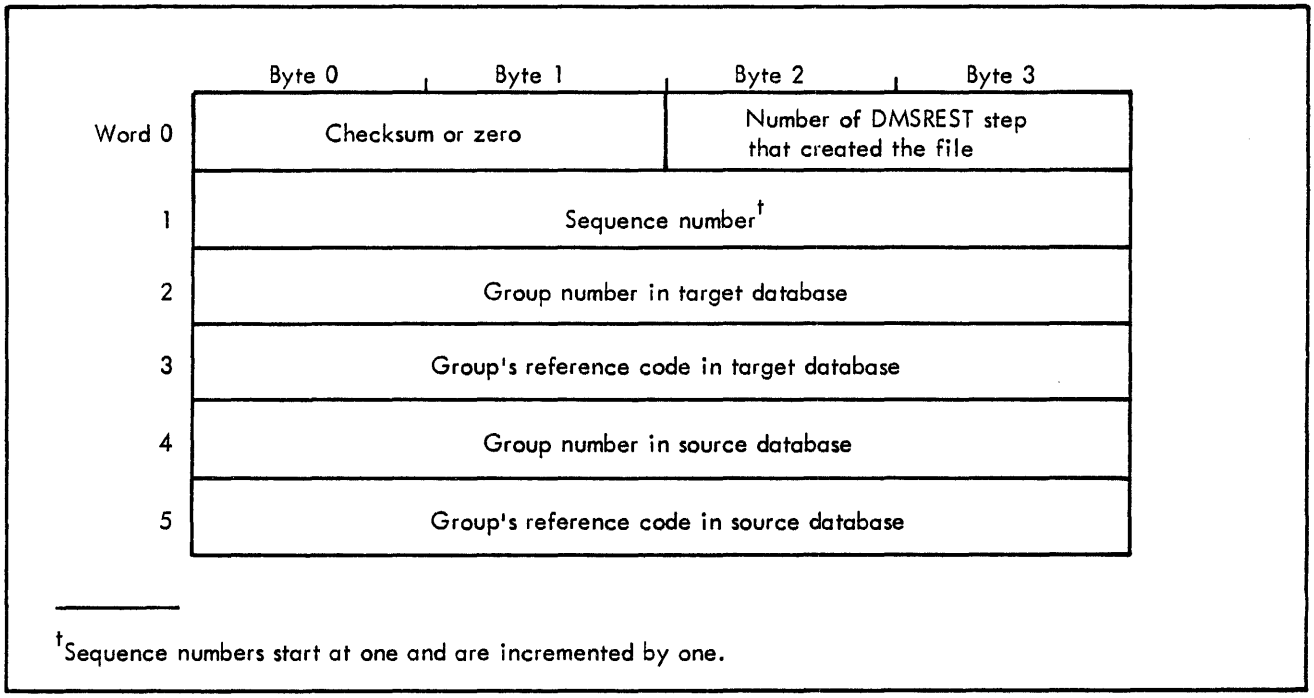


Figure K-1. Conveyed Group's Reference Code (CGRC) Record Format

INDEX

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

XEROX

Publication Revision Sheet

APRIL, 1976

CORRECTIONS TO THE XEROX EXTENDED DATA MANAGEMENT SYSTEM (EDMS) REFERENCE MANUAL
(Xerox 560 and Sigma 6/7/9 Computers)

PUBLICATION NO. 90 30 12C, December, 1974

The attached pages contain changes reflecting the B00 version of EDMS restructuring which is a non-supported product. All changes marked by revision bars are not in the standard product; they are only in the product released from the User's Group library.

Pages in the C edition (and C-1 revision package) that are to be replaced are: 9, 65-70, 73-74, 83-84, 85-88, and 91-94.

20424
2C478
Printed in U.S.A.

File No.: 1X33

XP82B, Rev. 0

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

☐ Learning ☐ Installing ☐ Sales
☐ Reference ☐ Maintaining ☐ Operating

Is the material presented effectively?

☐ Fully Covered ☐ Well Illustrated ☐ Well organized ☐ Clear

What is your overall rating of this publication?

☐ Very Good ☐ Fair ☐ Very Poor
☐ Good ☐ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your name & Return Address

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

CUT ALONG LINE

FOLD ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 59153 LOS ANGELES,CA 90045

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
5250 W. CENTURY BOULEVARD
LOS ANGELES, CA 90045

ATTN: PROGRAMMING PUBLICATIONS

FOLD ALONG LINE

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

23001, 2.5C379, Printed in U.S.A.

XP82, Rev. 0